

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select “Pages from: first page # to: last page #” on the print dialog screen



The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

PARAMETER BASED SYNTHESIS OF SIGN LANGUAGE

The primary objective of this project is to demonstrate synthesis of signs in American Sign Language with a finite number of parameters. The parameters in this method include 22 key locations, six location modifiers, six orientation indicators, 46 hand shapes, six predefined movements and seven possible end effectors. These parameters can be used in conjunction with a computer script and they maximize precision and versatility while minimizing development time and resources. The parameters for each sign are stored in an individual text file averaging approximately half a kilobyte.

The success of this project is demonstrated using UGS's Jack Software, Version 4.0. Jack is a tool for the animation of human movement. The human Jack is modeled using joint angle limits and inverse kinematics methods when moving any limb to a destination. A Python interface to Jack can take the English equivalent of the sign as an input, call its parameters and command Jack to sign.

A graphical user interface (GUI) was constructed using MATLAB 6.5 as a user friendly tool to help grow the database of parametrically coded signs. The GUI takes all the parameters identified for each sign and prints them to a file. The GUI is easily expandable and new parameters will be implemented in the future. The parameters of these signs will be used in the future as the foundation for sign recognition technology. Animation of sign language is a significant step toward a fully integrated ASL/English translator.

PARAMETER BASED SYNTHESIS OF SIGN LANGUAGE

**by
Amanda Irving**

**A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Biomedical Engineering**

Department of Biomedical Engineering

January 2005

APPROVAL PAGE

PARAMETER BASED SYNTHESIS OF SIGN LANGUAGE

Amanda Irving

Dr. Richard Foulds, Dissertation Advisor
Associate Professor, Biomedical Engineering, NJIT,

Date

Dr. William Hunter, Committee Member
Professor, Biomedical Engineering, NJIT

Date

Dr. Sergei Adamovich, Committee Member
Assistant Professor, Biomedical Engineering, NJIT

Date

BIOGRAPHICAL SKETCH

Author: Amanda Irving

Degree: Master of Science

Date: December 2004

Department: Biomedical Engineering

Education:

- Master of Science, Biomedical Engineering,
New Jersey Institute of Technology, Newark, NJ 2004
- Bachelor of Science, Interdisciplinary Mathematics: Electrical Science
University of New Hampshire, Durham, NH 2003

Funded by:
Grants from the
Pfeiffer Foundation
and
Millbank Foundation

TABLE OF CONTENTS

Chapter	Page
1 INTRODUCTION	1
1.1 Objective	1
1.2 Causality of Deafness	2
1.3 Technological Resources	4
1.3.1 TDD and Relay Services	4
1.3.2 Common Text Based Technologies	5
1.3.3 Common Video Based Technologies.....	5
1.3.4 The Impact of the Cell Phone	6
1.4 Human Resources	6
1.4.1 Translators	6
1.4.2 Lip Reading	7
1.4.3 Managing in the Work Place	8
2 AMERICAN SIGN LANGUAGE	10
2.1 Origins	10
2.2 Grammatical Structure and Lexicon	11
2.2.1 Sign Components.....	12
2.2.2 Stationary Signs	12
2.2.3 Non-Stationary Signs	13
2.2.4 Iconicity in Signs.....	14
2.2.5 Directional Signs	15

TABLE OF CONTENTS
(Continued)

Chapter	Page
2.2.6 The Influence of English on ASL	16
2.3 Signed Exact English (SEE)	17
3 AMERICAN SIGN LANGUAGE AS A SECOND LANGUAGE.....	19
3.1 Learning American Sign Language	19
3.2 Primitive Sign Language Depiction.....	20
3.2.1 Photo with written description	21
3.2.2 Video resources	21
3.3 Symbol Based Description of American Sign Language.....	23
3.3.1 Stokoe's Notation.....	23
3.3.2 ASCII-Stokoe Notation.....	24
3.4 Signing Avatars.....	25
3.4.1 Paula, DePaul University.....	25
3.4.2 TESSA, TExt and Sign Support Assistant.....	26
3.5 Parameter based engines.....	28
3.5.1 SignSynth, University of New Mexico.....	28
4 PRODUCING AMERICAN SIGN LANGUAGE	31
4.1 Jack Software	31
4.1.1 Scene	31
4.1.2 Figures.....	32
4.1.3 JackScript	33

TABLE OF CONTENTS
(Continued)

Chapter	Page
4.1.4 Locations in Jack.....	34
4.2 Previous Work with Jack.....	34
4.3 Methodology.....	35
4.3.1 Choosing Parameters.....	36
4.3.2 Sign Name.....	37
4.3.3 Handedness.....	37
4.3.4 Hand Shape.....	38
4.3.5 Locations.....	38
4.3.6 Offsets.....	40
4.3.7 Orientation.....	41
4.3.8 End Effector.....	41
4.3.9 Movement with no Hand Shape or Orientation Adjustment.....	43
4.3.10 Movement Created by Multi-Segmented Signs.....	43
4.3.11 Executing the Sign.....	44
4.3.12 Special Features.....	46
4.4 Recording Parameters.....	46
4.4.1 Handedness.....	47
4.4.2 Hand Shape.....	47
4.4.3 Hand Location and Offset.....	48
4.4.4 Movement.....	48

**TABLE OF CONTENTS
(Continued)**

Chapter	Page
4.4.5 Orientation.....	48
4.4.6 Multi - Segment Signs.....	49
4.4.7 Saving and Clearing Data.....	50
5 CONCLUSION	51
5.1 Discussion.....	51
5.2 Demonstration.....	55
5.3 Breadth of Functionality.....	57
5.4 Long Term Vision.....	58
5.4.1 Applications for the Hearing	58
5.4.2 Applications World Wide.....	59
5.4.3 Implications in the Field of Sign Language to English Translation.....	60
APPENDIX A - WORLD FINGERSPELLING CHARTS.....	61
APPENDIX B - MATLAB CODE FOR A GRAPHICAL USER INTERFACE	66
APPENDIX C - PYTHON CODE FOR JACK SIGNING PROGRAM	105
REFERENCES	148

LIST OF TABLES

Table	Page
4.1 Parameter Labels and Significance.....	36
4.2 Key Locations Associated to Sites.....	39
4.3 Key Locations Associated With Sites.....	40

LIST OF FIGURES

Figure	Page
2.1 Signs depicting professions: 1.'TEACH' 2.'er', 1.'LEARN' 2. 'er.'.....	12
2.2 Signs with motions: WHERE, OLD.....	14
2.3 Signs that mimic: LISTEN, BASEBALL.....	15
2.4 Signs incorporating the first letter of their English equivalent: Water, Live	17
3.1 User interface for Solina and Krapez' software.....	22
3.2 Stokoe Notation Example: Don't Know.....	23
3.3 Pictograph: Don't Know.....	24
3.4 Paula of DePaul University signing 'nice.'.....	25
3.5 TESSA, TExt and Sign Support Assistant.....	27
3.6 Signer equipped with motion capture equipment.....	28
3.7 User interface for SignSynth, 1999.....	29
3.8 SignSynth models, 2001.....	30
4.1 Coordinate system for placement of Jack's hand.	41
4.2 Graphical User interface for storing sign parameters.....	47
5.1 Picture Description for 'Belt'.....	55
5.2 Segment One of Sign 'Belt'.....	56
5.3 Segment Two of Sign 'Belt'.....	56
5.4 Jack Performing 'Belt'	57

CHAPTER 1

INTRODUCTION

1.1 Objective

The primary objective of this project is to demonstrate accurate synthesis of signs in American Sign Language using a finite number of descriptive parameters. Methods to this point have predominately employed time consuming and labor intensive strategies in pursuit of an accurate signing model. This method uses no motion capture techniques or video based stream to depict Sign Language. The parameters used in this method include 44 key locations, 6 location modifiers, 6 orientation indicators, 46 hand shapes, 6 predefined linear movements and seven potential end effectors. The parameters chosen form a base for all progress to come. These are the initial, most relevant parameters that can be used to describe a large library of signs.

The success of this project is demonstrated through UGS's Jack Software, Version 4.0. Jack is a three dimensional rendering of a human modeled from body measurements from a 1988 anthropometric survey of US Marines. Jack models human movement as accurately as possible by implementing joint angle limits and inverse kinematics methods when moving any limb to a destination. Jack was designed by the University of Pennsylvania for ergonomics studies, but can be programmed to depict a wide variety of tasks [1].

Animation in Jack can be controlled by using a built in Python interface. Python is a high level object oriented programming language that is very stable and gives high priority to visual cues to define its structure [2]. Python command sequences can be

written in simple ASCII text files and executed in any sequence. Animation of sign language is achieved by executing these files to control the movements Jack makes.

After a successful synthesis engine has been constructed to make use of these parameters, the secondary objective is creating new signs without requiring expensive equipment or excessive human resources such as trained signers. A graphical user interface (GUI) was constructed using MATLAB 6.5 to allow someone not specifically trained in Sign Language translation, the Python programming language or the MATLAB programming language to extract the necessary parameters from a written or pictorial description. Each sign has an individualized file that contains a record of all parameters associated with the sign. These individual text files average 600 bytes. The GUI will automatically generate a file compatible with Python code that contains all the parameter values. This decreases the time required to create a new sign for the dictionary and increases the number of people capable of creating new signs. The sign can be demonstrated by Jack almost immediately and the parameters can be altered through the GUI during the same session if necessary. The GUI is designed to easily incorporate additional parameters in the future.

1.2 Causality of Deafness

Although scientists involved in the Human Genome Project have identified approximately 50 'deaf' genes, there are many other causes of deafness that are not connected with a genetic disposition [3]. Deafness is possible at any stage in an individual's life. Diseases contracted by a mother during pregnancy can result in deafness in their newborn. Babies born to mothers who contract a venereal disease, or

abuse drugs and alcohol are not uncommon victims of congenital deafness. Asphyxia due to difficult deliveries or disease contracted in the womb can also lead to deafness in a newborn child [4].

German measles commonly causes deafness in young children. Diseases such as Alport Syndrome are contracted at birth, but symptoms affecting hearing do not become evident until mid-childhood. Aging commonly affects the quality of a person hearing, although hearing deteriorates at different rates across the spectrum of humans. Overexposure to loud music can accelerate the deterioration of the auditory organs over time, or a very loud noise like that caused by firecrackers, airbags or a large explosion can cause instantaneous damage to the auditory organs [4].

The length of time a person spends in the hearing world has a profound affect on how they communicate and think. Babies born deaf and diagnosed early commonly consider Sign Language their first and primary language. Speaking in English is usually very difficult and written skills are a challenge to develop. Hearing children learn the spoken form of English before they learn the written form, but deaf toddlers do not go through this learning sequence. Spoken and written English share very similar grammar rules, but American Sign Language and written English do not. This causes a handicap when children who have been deaf most of their life try to learn English. The average reading and comprehension level for juniors and seniors in deaf schools is 4th grade. People who develop language skills in English before losing their ability to hear retain these skills, think in English and are more proficient in written English [5].

1.3 Technological Resources

Many technologies designed to ease communication between hearing and deaf are text based. For Deaf users with proficient skills in written English this is not a great inconvenience, but for the congenitally Deaf who prefer not to communicate in English, these technologies fall short of their needs. Video and imaging technologies allow communication in almost real time using devices such as webcams or video phones, but these require both parties to know American Sign Language. A real time interface between a Sign dependant Deaf person and a hearing non-signer has yet to be constructed to fit the needs of all involved.

1.3.1 TDD and Relay Services

Telecommunications Device for the Deaf (TDD or TTY) and relay systems have been specifically developed to help the deaf communicate over the phone line. TDD allows hearing and non-hearing to communicate when each party is equipped with the TDD system. The system has a keypad and display screen very similar to today's common online instant messaging systems. Communicating over a TDD requires both parties to be equipped with the special device. Many hearing people do not own one and in a work environment, supplying clients with the TDD or requiring clients to purchase one is not a diplomatic or viable option.

A relay system allows a hearing person without a TDD talk to a hearing person who requires it and vice versa. All contact is assisted by a relay operator who is both hearing and has a TDD. The relay operator reads aloud the typed communication from the deaf person to the listener and types the hearing person's response. The use of a relay system eliminates the equipment dilemma but adds an unknown factor in the operator.

Privacy is non-existent in a conversation where a stranger is being relied on to translate. A deaf person would not use a relay operator to contact their credit card company where they require an account number, password, and other confidential information. The users of a relay system have no way of knowing the exact message given by the operator or of how the operator introduces and conducts themselves. Some managers refuse to use relay systems citing that they are too unprofessional for their work. While TDD and relay systems have some disadvantages, small companies may or may not even have these technologies available which can trap a deaf employee [6].

1.3.2 Common Text Based Technologies

Some common equal-opportunity communication technologies that are readily available and used for everyday communication are e-mail, text pagers, Instant Messaging software and closed captioning on television and movies. While these technologies can completely accommodate some, they do not offer real time communication that is important in collaboration and building relationships at work, school and in everyday life.

1.3.3 Common Video Based Technologies

Webcams allow transmission of video with only slight time delays. Two signers can communicate using American Sign Language without physically being in the same place. Webcams are readily available with affordable low-end models for anyone with a computer. Hardware quality and speed of Internet connection determine the quality of the video being received by the other party. Digital cameras often offer the capabilities of recording short video clips and digital camcorders are becoming more commonplace. This digital information can be emailed or streamed to others in the same manner text or

email is sent and received. Faster Internet speeds are becoming more widely available across the nation and technological advances drive down prices of existing hardware.

1.3.4 The Impact of the Cell Phone

A technology not available to the hearing impaired that has had an indirect effect is the cell phone. The cell phone is an important technology that brings to light the shortcomings of technologies to assist the deaf. It creates a new pressing demand for research in deaf technologies.

It is very common for hearing business people to take advantage of the cell phone to communicate with partners and clients while traveling out of the office. Traveling and doing business concurrently is more and more commonplace; it is having a great affect on how deaf employees can get ahead in fields where this is more prevalent. The reality of life is that communication complications create glass ceilings in some career paths [6].

1.4 Human Resources

Human resources are any resource that is non-material or technology based. Skills are developed by deaf people to help them interact with a primarily hearing world. There are also hearing people who have adopted skills and tendencies useful to deaf people.

1.4.1 Translators

Some hearing impaired people regularly use interpreters to facilitate communication. The most informal translators are hearing family and friends communicating in a setting where not everyone knows ASL. Deaf programs in hearing public schools are being established with more frequency. This commonly involves hiring a translator for the deaf

students in the program. These examples are either impromptu or institutionalized, but in either case, the cost, if there is any, is not commonly questioned.

In the office, and employer absorbs the cost of any interpreter or other service. Cost results in variability in accessibility to these services. In some companies, an interpreter will be available at important meetings, but is not regularly available for daily interactions. When an employee has an interpreter to assist in all daily communication, it is common for the interpreter to be an employee of the human resources department. In this situation the hearing impaired employee is not authorized to give their interpreter additional tasks when their language skills are not required. Having an interpreter also employed as a secretary or assistant, is generally thought of by deaf employees as an ideal working situation [6].

The quality and speed of interpretation is a very important concern. In all environments there is specific 'jargon' that needs to be understood to get a clean interpretation. Some interpreters are slow and incomplete in interpreting because of the fast pace or unfamiliar vocabulary. A deaf employee cannot always perceive these shortcomings and can experience difficulty giving input and suggestions in a rolling conversation [6].

1.4.2 Lip Reading

Deaf people often develop lip-reading skills to more easily understand what people say. A person who is able to hear volume changes and tone has the most success lip reading. Some phoneme sequences are impossible to distinguish from each other by lip position alone; tone and context are required to infer what the person is saying. Mumbling, facial hair and other factors can impede someone trying to lip-read, but over time lip-reading

individuals gets easier as familiarity develops. A solution to difficulties lip reading an unfamiliar person is to have someone familiar in the room to facilitate communication, which could be awkward if the meeting is about something sensitive. The understanding and willingness of hearing coworkers to adjust and be empathetic contributes to the success of the deaf worker. Sometimes just a positive attitude and willingness to enunciate can make communication and teamwork more effective [6].

1.4.3 Managing in the Work Place

Communication difficulties can lead to managers being less demanding of their deaf employees. Deaf employees can also become isolated and less likely to be promoted through lack of exposure or social contact. Some deaf employees acknowledge that working in a team environment is very difficult [6].

For deaf employees that are managers, obtaining information 'through the grapevine' is very important for obtaining casual feedback or intervening when problems between co-workers arise. While hearing people can discretely overhear other people's conversations, a deaf person lip-reading must be looking at the person speaking. An interpreter might not feel obligated to translate office chitchat, even when it is possible to communicate discretely. An asset to someone who is hearing impaired is a hearing friend or coworker that will inform him of this useful information [6].

Deaf personnel who own their own business or are elevated managers face the daily challenge of communicating with customers and training employees. A manager in charge of employees can choose to spend time teaching basic signs or inventing new signs related to the employees' jobs in order to facilitate communication. An owner of a retail shop does not have an opportunity to serve a static set of customers and educate

them individually. Even if this was a possibility, without a dynamic of customer base, sales will not grow and it is detrimental for the business. New customers can find the communication gap intimidating and may prefer to find the material they need in a more familiar setting [6].

CHAPTER 2

AMERICAN SIGN LANGUAGE

2.1 Origins

American Sign Language is accepted as the primary language in deaf communities throughout the United States and English speaking Canada. Thomas Hopkins Gallaudet was an American who traveled to Europe to learn about the European education of the deaf. Gallaudet studied French Sign Language (FSL) and asked Laurent Clerc, a Parisian FSL instructor to come to the United States. Gallaudet founded the first school for the deaf in Hartford, Connecticut in 1817. Schools for the deaf became more common and were modeled after the Hartford school. This initiated the standardization of what would become known as American Sign Language. It was common for these first schools to be boarding schools, allowing more deaf children the opportunity to learn. As the number of these schools increase, and as more hearing schools adopt programs for integrating deaf students, the older schools adjusted to the trend by reducing boarding opportunities or enrollment [7].

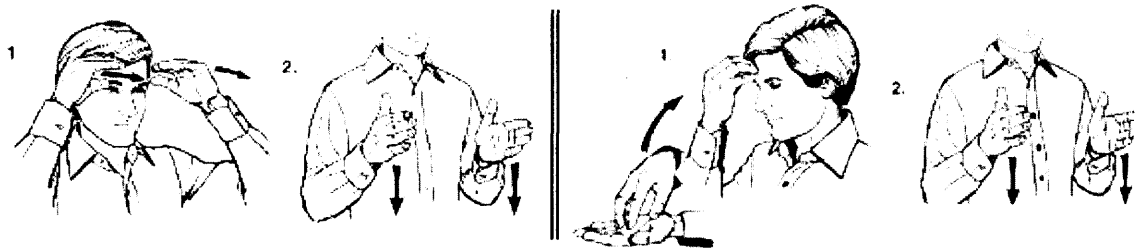
While French Sign Language is generally given credit for being the mother of American Sign Language, almost 200 years have passed since Clerc first arrived in the United States to educate the deaf. Both languages have evolved separately over 200 years and some similarities in the languages have faded.

2.2 Grammatical Structure and Lexicon

American Sign Language is unique from written and spoken English. This causes some confusion for people trying to learn ASL as a second language. American Sign Language is an object-oriented language, which means the object of the sentence is given highest significance by being signed first with the subject next, then the verb. In the example "I am taking the book," the sign for 'BOOK' is signed first, then 'I,' and 'TAKE' are signed in that order [8].

There are no ASL signs for the articles 'a,' 'an,' and 'the' in English. All forms of the English verb 'to be' (i.e. am ...ing, are, is, was) are disregarded and there are no group of signs distinguishing each verb tense of one single action. Time sequence is communicated by depicting specific occasions such as 'YESTERDAY,' 'LAST YEAR,' 'TOMORROW,' OR '10 O'CLOCK,' or more general time sequence such as 'BEFORE' and 'AFTER' [8].

Some prefixes and suffixes can be added to signs to change their meaning. A closed fist with thumb touching the bottom of the chin and continuing away from the body is used as a negation (i.e. 'I am unhappy,' 'I am not dancing'). A suffix exists to change an action into a profession (i.e. the 'er' equivalent in English). Adding this suffix creates new words: teach becomes teacher, learn becomes student and cook becomes chef [see Figure 2.1]. After the verb is signed, a transition, the 'ER' is signed by extending hands in front of the chest, palms facing in, and traveling down as to imitate a human silhouette [8].



http://library.thinkquest.org/10202/school_student.html
http://library.thinkquest.org/10202/school_teacher.html

Figure 2.1 Signs depicting professions: 1.'TEACH' 2.'ER', 1.'LEARN' 2. 'ER.'

2.2.1 Sign Components

Signs can be broken down into five kinetic components:

hand shape: the formation of the joints in the hand, not including wrist angles

location: the hand's position in space

orientation: the direction the palm is facing along with 'pointing' direction as if hand shape were a closed fist with index finger extended.

movement: a change in location or hand shape during a sign, including linear, circular, arced and repeated trajectories.

facial expression: any expression of the face used to describe or reinforce the concept being conveyed [9]

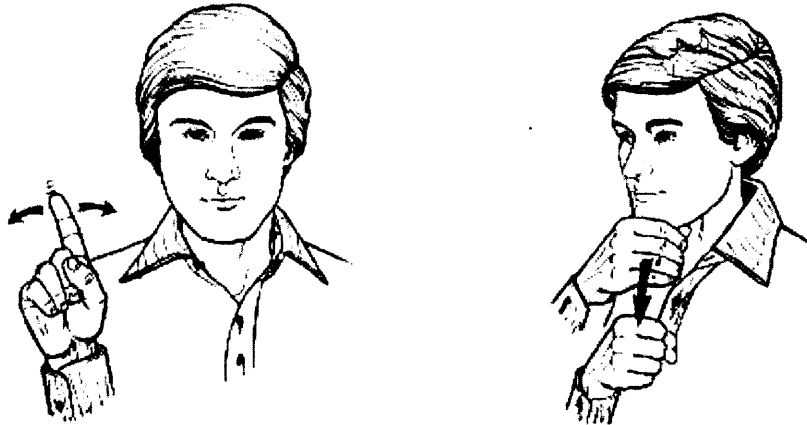
2.2.2 Stationary Signs

Stationary signs are associated with no distinct movement. Their hand shape, location, orientation and facial expression completely express the significance of the sign.

2.2.3 Non-Stationary Signs

Signs in ASL incorporate movement in a variety of ways. Seemingly subtle movements can completely change the meaning of some signs. For example, a closed fist with palm oriented "in" toward the body, and with the index finger extended upward is the sign for the number '1,' rotating the palm out and shaking the finger slightly like an inverted pendulum, results in the sign 'where?' In signs with circular motions, the size of the circle has significance. 'FAMILY' is one such sign where the size of the circular motion can describe the size of the family; a facial expression usually confirms the intentions when exaggerating the circle to signify a large family. 'IMPORTANT' is a sign that can also be modified by exaggerating the size of the circular motion. Increasing the size of the motion accompanied by an appropriate facial expression changes 'IMPORTANT' to 'VERY IMPORTANT' [8].

Signs with linear movements can be modified similarly. The sign for 'OLD' uses the 'O' hand shape starting at the chin. The length of the movement down and away from the body can indicate the intention to sign 'OLD,' 'VERY OLD,' or 'ANCIENT.' [See Figure 2.2] Speed is another way movement can modify signs. 'DEAF' can be signed by first pointing to the ear, then mouth; a deliberately quick motion can mean 'PROFOUNDLY DEAF.' Context and facial expressions clarify the intentions of the signer [8].



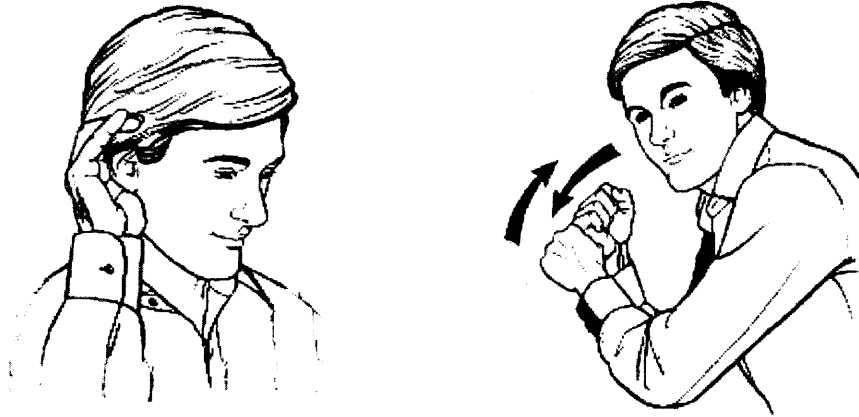
http://library.thinkquest.org/10202/basic_where.html
http://library.thinkquest.org/10202/basic_old.html

Figure 2.2 Signs with motions: WHERE, OLD.

Changes in hand shape are especially common when signing proper names and places. Other signs also have hand shape transitions: 'UNDERSTAND' involves an 'X' to 'l' transition and 'The Bronx' involves a 'B' to 'X' transition. These are only two examples of hand shape shifts during a sign [8].

2.2.4 Iconicity in Signs

Some signs resemble the object or action that it intends to convey. These are the types of signs that are more likely to be understood by cultures outside of the ASL community. 'DRINK' uses the 'C' hand shape and imitates the action by bringing an invisible cup up to the signer's lips. 'LISTEN' uses the 'C' hand shape to cup the ear, straining to listen. 'BASEBALL' is depicted by both hands in 'S' hand shapes as if holding a bat across the shoulder [See Figure 2.3] [8].



http://library.thinkquest.org/10202/verbs_listen.html
<http://library.thinkquest.org/10202/sport.html>

Figure 2.3 Signs that mimic: LISTEN, BASEBALL.

Some signs depict culture specific to the United States. The sign for 'MAN' is derived from tipping the brim of the signer's imaginary hat. This sign originated in a time where it was more common for men to wear hats. It is possible that the sign will evolve with time and reflect a more current ideal. Other signs express how Americans view other cultures; or express a common physical characteristic. A physical characteristic that stands out among Asian cultures is the shape of the eyelids and this has been incorporated into the sign. Some signs like this have evolved in order to become more politically correct or reflect America's most current view of other cultures [8].

2.2.5 Directional Signs

Some action signs that are regularly associated with an indirect object incorporate the subject and object into the sign. An example is 'TO ASK': the sign starts by pointing at the person doing the asking, and then the signer hooks his index finger while adjusting to point at the person being asked. This is a sweeping sign that should be completed in one motion. Actions that follow similar guidelines are show, give, pay, and tell [8].

2.2.6 The Influence of English on ASL

There is a single sign in ASL for each letter in the English alphabet. Using these signs in sequence to spell out words is called fingerspelling. Fingerspelling is not ASL but a standard tool used to bridge the two languages. Fingerspelling is regularly used for the proper names of people and places where appropriate signs do not exist. New signers often use fingerspelling while they are learning the ASL equivalents, but relying on fingerspelling to communicate is not efficient and makes the false assumption that all ASL signers are fluent in written English [8].

Many popular places and people have signs that incorporate the hand shapes associated with the letters in their English name. Signs for local places contribute to the local dialect and may not be known nationwide. In the Northeast, the deaf community of New York City has individual signs for each of the boroughs and popular neighborhoods like Greenwich Village [8].

Some signs come in groups with similar location and movement characteristics, but differ by hand shape to reflect the English word they are describing. 'RESTAURANT' is signed by tapping the right then left sides of the chin with an 'R' hand shape. Changing the hand shape to 'C' while using the same motion creates 'CAFETERIA.' Another family of signs is 'TEAM,' 'CLASS,' and 'FAMILY,' that use the 'T,' 'C,' and 'F,' hand shapes respectively. Other signs that incorporate the first letter of the English equivalent are 'WATER' and 'LIVE.' Water is signed by using the 'W' hand shape and motioning the signer's index finger to his lower lip. Live (as in 'I live in this house') is a symmetric sign using the 'L' hand shape, thumbs pointed up with an upward

motion by both hands [See Figure 2.4]. Incidentally, using an 'A' hand shape with the same motion as 'LIVE' expresses 'ADDRESS,' as in mailing address [8].



http://library.thinkquest.org/10202/misc_water.html
http://library.thinkquest.org/10202/basic_live.html

Figure 2.4 Signs incorporating the first letter of their English equivalent: Water, Live

2.3 Signed Exact English (SEE)

Signed Exact English (SEE) uses gestures to demonstrate English grammar with signs. ASL signs are used in SEE, but SEE incorporates English grammar in the sign sequence. There are signs depicting all articles and suffixes such as, 'ing' and 'ed' are added to the action sign that convey time sequence and tense information [10]. In American Sign Language there is no differentiation between 'I' and 'ME,' but in SEE a new sign is invented for 'I.' Signed Exact English evolved in the 1970's as a tool to enforce English grammar in deaf classrooms [11]. It became evident that deaf students were not graduating high school with a proficiency in written English and SEE was the proposed solution. SEE is used today in varying prominence. Some teachers use SEE only in English grammar classes, but some teachers use SEE in all subjects. A side affect of the

development of Signed Exact English is that some dictionaries [12] and other literature publish signs used in SEE that do not exist in ASL. Many times these signs are defined without informing the reader of the proper use.

CHAPTER 3

AMERICAN SIGN LANGUAGE AS A SECOND LANGUAGE

3.1 Learning American Sign Language

Learning any language poses distinct challenges. Some have unfamiliar written characters, and others have phonemes unfamiliar to the tongue. These difficulties are not encountered learning ASL because it involves neither writing nor speaking; ASL has its own set of difficulties.

ASL occupies the three-dimensional space around the signer's body. All the features of the language are important to learn correctly, because subtle changes in one characteristic can have subtle or drastic change in the meaning. Changes in orientation have a great effect on signs. The letters 'K' and 'P' have identical hand shapes; they vary only by orientation. 'K' requires the index finger to point up; 'P' requires the index finger to point downward and away from the signer's body. In some signed expressions, facial expression indicates sarcasm, humor, or that a question is being asked. When asking someone's name, the sign sequence WHAT-IS-YOUR-NAME is not appropriate. In an informal setting, signing 'NAME - YOU,' while using facial expressions to convey the concept of inquiry. In oral languages changing statements to questions can be achieved by an inflection in voice tone; without the benefit of this information, ASL signers rely on visual cues like magnitude and speed of arm movements along with facial expressions [8].

Video and books containing two dimensional illustrations or photographs are the standard products purchased with the intent of self-teaching ASL. Tom Humphries,

author of *Learning American Sign Language* and Lecturer at the University of California, San Diego uses the forward in his book to offer an important insight on using books to self-teach:

You should be aware that this book is not intended to be self-instructional. No book can be truly self-instructional when the objective is to learn a language that uses gesture and vision. However, the illustrations and exercises in this book will help you recall and practice what your teacher has presented in class or what you have seen on practice videotapes. This book will serve a purpose for which textbooks are ideally suited – as a resource and reference for your ongoing study when no model of American Sign Language is present to answer your questions [13].

The most practical and efficient way to learn ASL is to attend a class instructed by a native signer with students at a similar ability level. In larger cities, private and public companies and institutions teach ASL classes, but availability may be less in areas where the deaf population is less dense.

3.2 Primitive Sign Language Depiction

Resources are becoming more readily available for learning American Sign Language or translating from English to ASL. Regardless of effectiveness, these methods are not compatible with the type of input required for a sign synthesizer program. A new method

must be developed for describing the parameters of a sign without relying on photographs, video and verbose text descriptions.

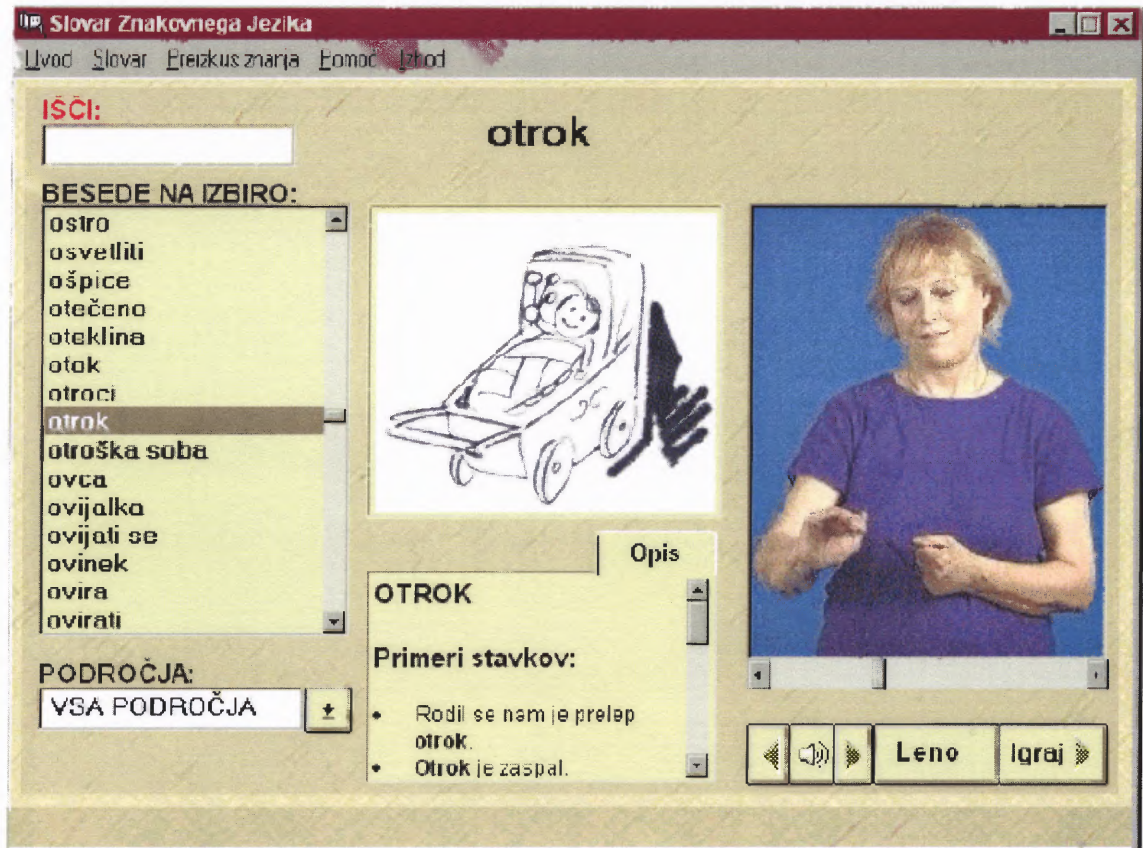
3.2.1 Photo with Written Description

The most primitive English to American Sign Language description is a stationary picture. Sometimes these stationary pictures that appear in ASL dictionaries include arrows describing the motion of the sign, or a written description of the motion. The Gallaudet Survival Guide to Signing contains up to three signs per page with a written and illustrated description of the sign's formation and movement [14]. In the development of any sign dictionary, quality is often sacrificed to increase quantity of product or reduce production costs.

3.2.2 Video Resources

An alternative to illustrations and photographs is video. While video is still two dimensional, it offers real time movement and eliminates the confusion that inserted arrows can create. Martin Sternberg produced the first English-ASL translator using video clips to depict sign sequences. The software has been available commercially through Harper Collins since 1994. Sternberg's electronic translator disregards the grammatical rules of American Sign Language, it simply signs the user input word by word [15].

Two Czech gentlemen, Franc Solina and Slavko Krapez followed up with Sternberg's work by producing a Czech Sign Language dictionary on CD-Rom. Solina and Krapez included video clips of the signs along with an illustration of the physical object, if the word could be depicted as a picture [See Figure 3.1] [16].



<http://eprints.fri.uni-lj.si/archive/00000039/01/MMdict2001.pdf>

Figure 3.1 User interface for Solina and Krapez' software.

Solina and Krapez were forced to reduce the frame rate of their movies to 15 frames/second in order to fit all the digital information on one CD (compared to 30 frames/second that is standard for television in the United States). The largest contribution was the blending algorithm [17] developed to reduce awkwardness in the transition of combined video clips. Each sign filmed started at a neutral position and ended in the same neutral position. A vector from the signer's elbow to wrist was used to blend adjacent signs. Frames were removed from the end of the first and the beginning of the second clip to minimize the awkwardness of the elbow to wrist transitions [16]. While this is an improvement on previous methods, it still results in rough transitions.

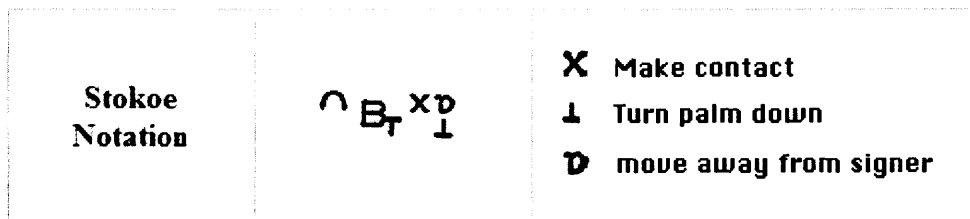
Fluid signing contains no neutral position for the signer to start from and return to after each sign. The initial position of a sign is inherited from the last sign and the body unconsciously makes that transition.

3.3 Symbol Based Description of American Sign Language

There have been notable attempts to methodically describe ASL signs using only symbols. The two methods of consequence for this research are Stokoe's Notation, and a linear text based adaptation of this technique called ASCII-Stokoe Notation.

3.3.1 Stokoe's Notation

William Stokoe's work in analysis of the biomechanics of American Sign Language (ASL) was a breakthrough in the understanding of the phonetics of the language. Stokoe created a notation that allows many signs in ASL to be described using 55 symbols he associated with the hand shape, location or movement characteristics of the sign [9].

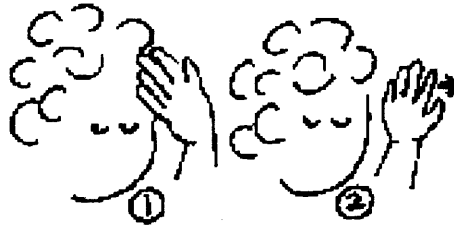


<http://www.signwriting.org/forums/linguistics/ling016.html>

Figure 3.2 Stokoe Notation Example: Don't Know

Figure 3.2 depicts an example of a sign in Stokoe Notation. The concave arc indicates the sign starts in the upper half of the face, the B indicates the hand shape is the fingerspelled B and the T subscript designates that the initial orientation is palm down. Stokoe's notation uses a two dimensional format to describe signs with multiple motions or independent motions for the dominant and non-dominant hands [9]. Movements are

executed one column at a time. X indicates that the first motion is contact with the forehead and the upside down A and T indicate the second motion is rotating the wrist down and away from the body. Comparing to the pictograph below [Figure 3.3], it is inconclusive which is more accurate or descriptive.



http://www.lesstutor.com/ees_asl_negative_phrases.html

Figure 3.3 Pictograph: Don't Know

Regardless of which method is thought to be better or more scientific, neither is convenient for inputting into an automated program. The symbols used in Stokoe's notation are very descriptive but are not necessarily compatible with the traditional ASCII symbols, which are standard keyboard inputs to a computer program [9].

3.3.2 ASCII-Stokoe Notation

Mandel has worked on a solution to the transliteration challenge from Stokoe's notation of signs to a linear ASCII based solution. Mandel incorporates the three parameters of biomechanical movement developed by Stokoe: location, hand shape, and movement. Mandel contributes a methodology to express Stokoe's symbols in a linear sequence of ASCII characters. Mandel uses slashes (/) to separate the parameters allowing for multiple movements to be represented in sequence. Using a comma and adding parameters to the linear sequence can account for sequences of movement [17]. The

SignSynth project actively uses ASCII-Stokoe notation, though its implementation is not disclosed.

3.4 Signing Avatars

3.4.1 Paula, DePaul University

DePaul University has been developing an animated figure named Paula to convey messages in ASL. Paula is an animated character with enlarged hands and facial features [See Figure 3.4]. The designers of Paula have made detailed facial expressions a top priority in the quest to create a believable character. The project involves 16 current faculty members and students and an estimated 25,000 hours have been spent developing the technology over 5 years [18].



<http://asl.cs.depaul.edu/demo.html>

Figure 3.4 Paula of DePaul University signing 'nice.'

Paula uses speech recognition as the primary input to the animated interpreter. The creators have decided to focus on developing Paula's vocabulary specifically to

communicate messages in an airport setting where important messages are not necessarily appropriate for written warnings. While Paula might be very effective in getting a message across, programming her gestures and facial expressions is very labor intensive.

3.4.2 TESSA, TExt and Sign Support Assistant

TESSA is a collaborative project in Britain to assist Deaf postal customers [see Figure 3.5]. TESSA is an avatar who signs in British Sign Language (BSL) as commanded by the teller at the Post Office. The teller uses speech recognition to indicate a question to the Deaf customer, and then TESSA signs the question on a monitor available to the customer. The teller is only able to communicate phrases and sentences that are in the database of signs, which only relate to post office functions. Another handicap is that after the teller has used TESSA to sign his or her question, the deaf customer must communicate a response. Presumably the teller is not proficient in British Sign Language and the deaf customer's response is a combination of miming or handwritten response [19].



<http://www.sys.uca.ac.uk/SysResearchWeb/groups/image%20%26%20signal%20processing/Tessa/Tessa.html>

Figure 3.5 TESSA, Text and Sign Support Assistant.

Motion capture techniques are the basis for TESSA's realistic motions. A proficient signer in BSL is equipped with head gear, vest, arm bands and gloves [see Figure 3.6]. This apparatus records joint angles, positions in space and velocities which are stored and used to manipulate the avatar. The facial expressions animated by TESSA are the actual facial expressions of the signer who programmed the sign. Reflective dots can be seen on the signer's face in Figure 3.6 which are used to extract information about key points on the signer's face. These key points are used to illustrate facial expressions on the avator. Every time words or phrases are added to the database of potential signs, the equipment must be set up and the translator must travel to the command center for the session [19].



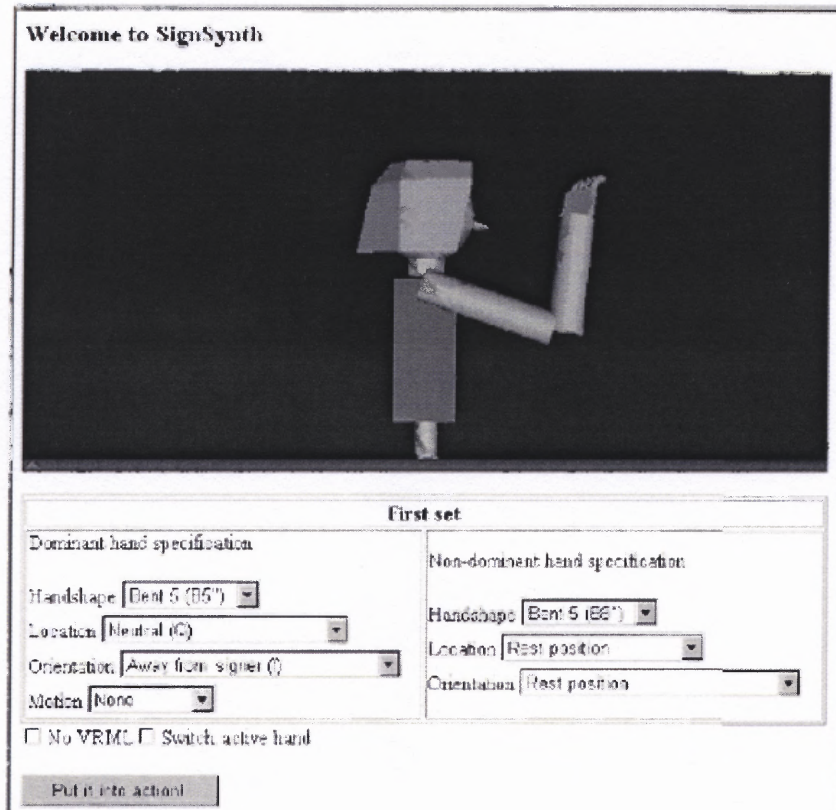
http://www.bbcworld.com/content/clickonline_archive_35_2002.asp?pageid=666&co_pageid=3

Figure 3.6 Signer equipped with motion capture equipment.

3.5 Parameter based engines

3.5.1 SignSynth, University of New Mexico

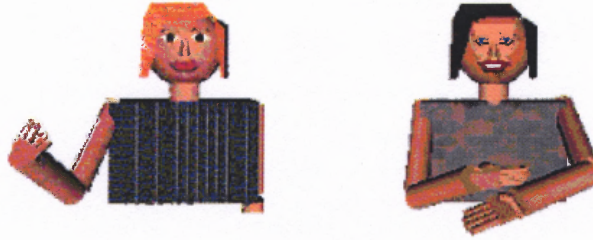
A recent surge in interest has increased the current number of projects in the sign language synthesis field. Angus B. Grieve-Smith of the University of New Mexico has developed a program called SignSynth that manipulates a primitive humanlike figure with cylindrical arms and basic geometric shapes making up its anatomy. SignSynth follows the principle that every sign can be defined by its hand shape, location and movement while adding the additional parameter of orientation. A version was presented at the 1998 First High Desert Student Conference in Linguistics [See Figure 3.7] [20] [21].



<http://www.unm.edu/~grvsmth/portfolio/sssp.pdf>

Figure 3.7 User interface for SignSynth, 1998.

SignSynth allows for user inputs in the form of pull down menus. For the dominant hand, the inputs are: hand shape, location, orientation, and movement. For the non-dominant hand, the GUI displays only hand shape, location and orientation inputs. The robotic figure above displays a hand in the 'bent 5' hand shape with left hand at its side [21].



<http://www.unm.edu/~grvsmth/signsynth/>

Figure 3.8 SignSynth models, 2001.

The female figures [Figure 3.8] represent the advancement in technology after three years on this project. In these three years the developers of SignSynth created a more visually pleasing model by adding color and texture, but the crude geometric shapes are the same as the original model. The demo associated with the 2001 model is available with no cost from the University of New Mexico, but requires the user to input the parameters of the sign in ASCII-Stokoe notation for the figure to appear. There are no example notations or other tutorials available to assist someone using the demo. The data structure for this project is not disclosed in the available documentation [22].

CHAPTER 4

PRODUCING AMERICAN SIGN LANGUAGE

4.1 Jack Software

UGS's Jack software is proficient in creating humanoid three-dimensional graphical figures capable of most human biomechanical movements. This software was originally designed by the University of Pennsylvania for ergonomic study, and used anthropometric surveys from the 1988 survey of US Marines. It can be manipulated to perform ASL with the same success as it can be manipulated to do the activities in ergonomics analysis [1].

4.1.1 Scene

A global coordinate system exists that describes the scene where Jack is located. The X and Z axes of this scene are orthogonal vectors, which describe the floor of the scene. The Y axis runs vertically, normal to the floor. All movements and rotations of individual objects are with respect to this global coordinate system unless otherwise specified, so it is important to understand and work with this coordinate system.

The view of the scene can be adjusted through mouse action and keyboard shortcuts. This is an important aspect of the program because it allows the user to view Jack at all angles, without interfering with his actions. Jack can be viewed from the front, side, and back, but more interestingly can be viewed from above, a feature not offered by traditional signing media.

4.1.2 Figures

Human figures can be chosen from Jack's library of standard and default body types, but it is also possible to modify the length and weight of each body part manually through Jack's customizing feature. The default male and female body type are used in this method. Other figures such as boxes, chairs, and other furniture are among the figures that can be created to exist in the scene with Jack although boxes scaled to the needs of the project are the only non-human figures used in this method.

A box is one of the most basic figures available in Jack. Regardless of the size of the box, it is considered one segment and its orientation is defined by a coordinate system originating at one corner of the box and extending along the three intersecting edges of the vertex.

Jack is a much more complex figure and is made up of 70 segments. A joint is defined as an intersection of two segments, thus Jack contains 69 joints. The size of each segment differs and important locations on each site are designated with descriptive handles. For a small segment, like most bones in the fingers, there are only two sites described: the distal end and the proximal end. The right palm is a larger segment and its importance is reflected in the 13 sites identified. Each segment, joint and site is identified with a descriptive handle and its own coordinate system. These handles allow a programmer to manipulate or extract information about the body part and the existence of a local coordinate system allows objects to move in relation to the axis of the site, as opposed to the coordinate system of the scene.

4.1.3 JackScript

JackScript is a group of built in functions designed to manipulate Jack. JackScript is written in Python code, which is the primary reason that the Python programming language was chosen to control all of Jack's movements. Jack 4.0 contains consoles in Python, TCL and LSP programming languages that provide an interface to controlling Jack's movements. The JackScript codes and Python scripts are executed through the Python console.

A brief overview of some functions is supplied to better understand the methods:

GetLocation() - input is an object and function returns 4x4 transformation matrix representing base coordinate system of object.

Trans() - input is a 4x4 transformation matrix and function returns translation array from scene's coordinate system.

XYZ() - input is a 4x4 transformation matrix and function returns Euler rotation angles from scene's coordinate system.

AttachTo() - input is an attachment site, function causes the location of the object to be attached to coincide with the attachment site.

MoveTo() - moves an object to inputted 4x4 transformation matrix.

Move() - moves an object by altering translation and/or rotation. AsSeenBy input identifies the set of axis that movement is relative to; the scene's coordinate system is the default axis.

Reach() - moves hand to inputted destination. The 'endeff' option identifies a point on the hand designated to align with destination. The 'otype' option designates how the end effector aligns with the destination.

4.1.4 Locations in Jack

Jack stores all types of location descriptions in a four by four matrix incorporating both rotation data and translation data. The 'trans' function extracts translation values from the four by four matrix and the 'xyz' function extracts the Euler rotation angles. The values extracted are relative to the global coordinate system and are expressed in radians and centimeters by default. The expression $xyz(\theta_x, \theta_y, \theta_z)*trans(\Delta_x, \Delta_y, \Delta_z)$ is the alternate notation for the complete 4x4 transformation matrix. The xyz component is extracted from orientation information and the trans component is extracted from location information; it is not necessary that these components originate from the same source to combine for a new transformation matrix.

4.2 Previous Work with Jack

Work by Jerome Allen introduced the idea of 'virtual cubes' to act as destinations used by the Reach function. A scene was built with the Jack figure and individual cubes at all desired locations in all desired orientations for the existing signs. These figures were saved in an .env file for recollection at a later time. This method requires a mechanism to add new cubes when new destinations and orientations are required. The visibility of all cubes was turned off so as not to distract the viewer watching Jack [23].

Allen created end effectors on the tips of the four fingers on each hand and uses the tip of the index finger as the end effector for the demonstrated signs. Another parameter that was used was a wrist angle designated in conjunction with the Reach destination alignment. The wrist angle described the relationship between the wrist and forearm, but did not influence to the relationship between the hand and Jack's body.

Using the tip of the finger is extremely convenient for some signs, but the hand shape should involve a finger that is extended to be most effective. Also, using the most distal site of the hand as an end-effector can lead to 'sloppiness' in the more proximal joint angles as Jack aligns itself with the destination [23].

Allen also takes advantage of the Python interface that Jack supplies. The user writes a sentence to be signed and each word is checked against the dictionary of available signs. If the word is in the dictionary, Jack executes a sequence designed for that sign; if the word is not in the dictionary, Jack fingerspells each letter in the word individually. One drawback of this procedure is that each sign has its own Python function that executes. This results in an extensive file that must be hand written to contain the functions for each sign. Adding signs to the inventory requires more than average expertise in Python and JackScript and current signs only involve movement of the right hand [23].

4.3 Methodology

The concept of parameterization is unique to this method. When adding or removing signs, there is no need to modify the core program whatsoever, or change anything related to the Jack software. Each feature of a sign is described in a small individualized text file which is independent of the software program used for animation.

The other feature unique to this method is the mathematical calculations of locations in real time as they are needed for animation. In Allen's method, locations were created in mass for each sign which cluttered the workspace and created technical complications for improving and adding to the sign base.

4.3.1 Choosing Parameters

Using Stokoe's 55 symbols to describe each sign is a method that is difficult to implement. Entering this notation into a computer program is not a straightforward task. Programs that try to follow this strategy commonly use Mandel's ASCII-Stokoe Notation as an input to their engine, but ASCII-Stokoe notation requires study to become proficient in using and would require an inventive program to extract the key information.

Table 4.1 Parameter Labels and Significance

Parameter Label	Significance
number	Quantity of motion segments to define sign
nondom	Non-Dominant hand tendency
handed	Dominant hand tendency
sign_name	Name of sign
DomShape	Hand shape of dominant hand
NonShape	Hand shape of non-dominant hand
DomLocale	Start location of dominant hand
NonLocale	Start location of non-dominant hand
DomEffector	Point on dominant hand serving as end effector
NonEffector	Point on non-dominant hand serving as end effector
DomX	Yaw angle of wrist from initial position for dominant hand
DomY	Roll angle of wrist from initial position for dominant hand
DomZ	Pitch angle of wrist from initial position for dominant hand
NonX	Yaw angle of wrist from initial position for non-dominant hand
NonY	Roll angle of wrist from initial position for non-dominant hand
NonZ	Pitch angle of wrist from initial position for non-dominant hand
DomOffX	Horizontal offset from DomLocale
DomOffY	Vertical offset from DomLocale
DomOffZ	Depth offset from DomLocale
NonOffX	Horizontal offset from NonLocale
NonOffY	Vertical offset from NonLocale
NonOffZ	Depth offset from NonLocale
DomSet	Set of movement descriptors for dominant hand
NonSet	Set of movement descriptors for non-dominant hand

This project focuses on seven characteristics for every sign. Hand shape, location, orientation, movement, endeffector, number of segments needed to completely describe the sign and the word or words representing the English equivalent. This information is stored in 24 parametric variables, depending on the number of segments being used to describe the sign. The parameters were chosen to achieve the highest balance between detail and ease of implementing the parameters in Jack's code.

4.3.2 Sign Name

The sign name is the English description of the sign in the shortest number of words. In the case where English synonyms can accurately describe the same sign, two files with different names are created containing identical data with the exception of the sign name. The sign name also acts as the name of the file containing the information for that sign, so spaces in the file name are not a viable option. In the case that the sign can not be described with only one word, there is a mechanism for the sign file to be saved as a word sequence with underscores replacing spaces.

4.3.3 Handedness

Jack is right handed, but the program was constructed to allow the possibility of a left handed Jack in the future. Signers have a natural inclination to be left or right handed and these signers would appear as mirror images when signing the same messages. For a person learning sign language, it could be useful to face someone signing in mirror image.

4.3.4 Hand Shape

The characteristics of Jack's hands are particularly important in this project. Jack has 15 joints, representing 20 degrees of freedom, fully capable of forming the primary hand shapes used in ASL. Jack has several standard hand shapes already built into its directory, but more importantly, Jack has a tool to facilitate saving new hand shapes for future use. The tool allows three joints for each of Jack's fingers to be adjusted; naming and saving this new hand shape builds a library of hand shapes. Allen has added over thirty-five hand shapes to the Jack library for use in communication in ASL [23]. A few additional hand shapes have been added to create a more complete library. JackScript's 'HandShape' function sets the hand shape of Jack's left and right hands by recalling saved hand shape data.

4.3.5 Locations

Forty-four sites were chosen to use as key locations for sign language. Twenty-six cube figures are attached to key locations on the body for ease of extracting location coordinates [See Table 4.2]. The cubes have descriptive handles for ease in programming and technical reading. The user can select one of these coordinates as a key location for the sign, or they can designate an offset from that point in any direction. These cubes are hidden so as not to distract the viewer.

Table 4.2 Key Locations Attached to Sites

Object Name	Attached To
r_orient	j.segment.right_palm.f11
l_orient	j.segment.left_palm.f11
r_index	j.segment.right_finger02.base0
l_index	j.segment.left_finger02.base0
r_ring	j.segment.right_finger22.base0
l_ring	j.segment.left_finger22.base0
r_pinky	j.segment.right_finger32.base0
l_pinky	j.segment.left_finger32.base0
r_middle	j.segment.right_finger12.base0
l_middle	j.segment.left_finger12.base0
r_thumb	j.segment.right_thumb2.base0
l_thumb	j.segment.left_thumb2.base0
r_cpalm	j.segment.right_palm.palmcenter
l_cpalm	j.segment.left_palm.palmcenter
cube_r_bpalm	j.segment.right_palm.front
cube_r_ear	j.segment.bottom_head.right
cube_l_ear	j.segment.bottom_head.left
cube_r_eye	j.segment.bottom_head.right_eyeball
cube_l_eye	j.segment.bottom_head.left_eyeball
cube_m_chin	j.segment.bottom_head.menton
cube_r_bicep	j.segment.right_upper_arm.front
cube_r_elbow	j.segment.right_lower_arm.front
cube_l_elbow	j.segment.left_upper_arm.distal
cube_r_wrist	j.segment.right_lower_arm.distal
cube_l_wrist	j.segment.left_lower_arm.distal
cube_neck	j.segment.neck.front

In order to incorporate more key locations eighteen cubes were placed in the scene and moved to the desired location. These cubes were moved from a nearby site located on Jack [See Table 4.3].

Table 4.3 Key Locations Associated With Sites

Object Name	Associated With	Offset
cube_r_hip	j.segment.l1.distal	(-12, -20, 15)
cube_l_hip	j.segment.l1.distal	(12, -20, 15)
cube_hip	j.segment.l1.distal	(0, -20, 15)
cube_r_shoulder	j.segment.l1.distal	(-12, 20, 12)
cube_l_shoulder	j.segment.l1.distal	(12, 20, 12)
cube_shoulder	j.segment.l1.distal	(0, 20, 12)
cube_chest	j.segment.l1.distal	(0, 0, 15)
cube_r_chest	j.segment.l1.distal	(-12, 0, 15)
cube_l_chest	j.segment.l1.distal	(12, 0, 15)
cube_nose	j.segment.bottom_head.sight	(0, -4, 3)
cube_m_mouth	j.segment.bottom_head.sight	(0, -8, 0)
cube_r_mouth	j.segment.bottom_head.sight	(-2.5, -6, 1)
cube_l_mouth	j.segment.bottom_head.sight	(2.5, -6, 1)
cube_l_brow	j.segment.bottom_head.sight	(2.5, 4, 1)
cube_r_brow	j.segment.bottom_head.sight	(-2.5, 4, 1)
cube_c_brow	j.segment.bottom_head.sight	(0, 4, 1)
cube_r_chin	j.segment.bottom_head.sight	(-2.5, -8, 1)
cube_l_chin	j.segment.bottom_head.sight	(2.5, -8, 1)

The cubes are associated with key locations, but not attached, serve as pointers that have more descriptive names than the original location. Cubes that are attached to body parts move with the body parts. The cubes only associated with key locations do not move when the body part moves. For this reason, these cubes are only associated with locations on the head and chest which are not expected to move significantly.

4.3.6 Offsets

The key locations indicated in the section above are at most, only several centimeters away from Jack's body. More locations must be available to allow signs to exist within the entire reach of Jack's arms. After a key location has been indicated, a distance away from the location can be defined in any direction. The distance is defined with reference

to the global coordinate system. The offset values are used to create an array which is then added to the translation array of the key location. The array calculated is used as the translation component of a new transformation matrix reflecting the goal destination.

4.3.7 Orientation

The destination cube must be rotated to reflect the parameters of the sign. This is done by using the 'Move' command which designates angles to rotate the cube as well as distances to move the cube. The default axis of rotation is the global coordinate system which is not convenient for rotating these small cubes. The `asSeenBy` feature of the Move commands allows rotation about the local coordinate system of a specified figure. The sequence of rotation is first, about the normal vector to the axial plane (yaw), next about the x-axis, effectively up and down (pitch), and lastly, rotation about the y axis, twisting the wrist (roll).

4.3.8 End Effector

The default in the Reach command designates the center of the palm as the end effector. This means that the center of the palm is aligned with the object at the destination. As an alternative, the base of the index finger has been designated in this method as the primary end effector [See Figure 4.1].

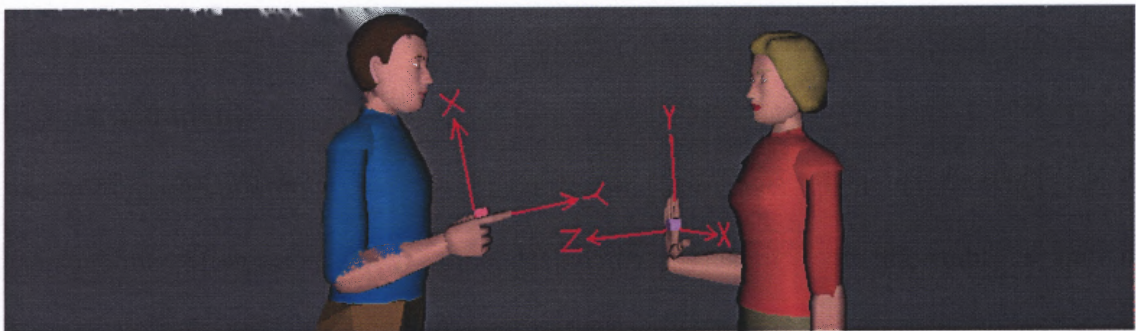


Figure 4.1 Coordinate system for placement of Jack's hand.

The reason for this choice is that one and only one axis is aligned in the 'pointing' direction, and one axis is in the direction normal to the palm with the hand shape selection being able to alter the intended pointing direction. A complication that arises when using the tip of a finger is that not all hand shapes have an extended finger to use for an accurate alignment. If the end effector were the tip of a finger, a bent finger could alter the intended palm direction. The ends of the fingers as end effectors are very important tools and using them increases the number of signs that can be successfully executed. This is especially true of signs that do not have a specific location to execute within and are primarily defined by relative positions and orientations between the hands. There are seven alternate end effectors that the user can choose: the center of the palm, the first knuckle, and the most distal phalange of each of the four fingers and thumb. Choosing the ends of the fingers is not recommended for signs where the specific finger is bent.

An advantage to using a point on the palm as the standard for orientation is that there are no joints in the hand more proximal. This means that when aligning the coordinate systems, all joints will be more distal and will not be altered when the hand is aligned with the goal cube. Using a more distal location can result in mutation of the hand shape as Jack does its best to align itself with the goal cube. To reduce mutation of the hand shape as the end effector aligns with the destination, this program reconfigures the hand shape after the movement to correct any potential mutation.

4.3.9 Movement with no Hand Shape or Orientation adjustment

Once a goal cube is in a specific orientation, it can be translated with no rotation. This allows user to specify relative movements such as: up, down, right, left, forward, and backward, without changing or repeatedly specifying the orientation of the hand. More than one movement can be selected, so if hand shape and orientation are not intended to change, creating a movement that moves diagonally up and to the right, or any other combination of movements is an option. Movements are accounted for in a similar way to the offset. After the initial position is calculated and recorded, an additional variable is created to store the location after the movement has taken place. The new variable reflecting the post movement location starts at the initial location, but for each movement, a 20cm offset in the appropriate direction is added to this value. The orientation does not adjust from initial to final position.

4.3.10 Movement Created by Multi-Segmented Signs

For signs that contain hand shape or orientation changes, using multi segmented signs offers a more natural looking sign. The hand shapes of Jack transition evenly in time when two locations are indicated for the sign. When the linear movement feature is used handshape1 is preserved throughout the movement, and ends with a fast transition to handshape2. Segment one reflects the initial position of the sign, then up to four more segments can define intermediate locations to the completion of the sign. The potential for defining intermediate locations is very important because Jack reaches for these cubes in an internal inverse kinematic algorithm and control is lost over how it gets to the desired location. If the path is very awkward and/or breaks the boundaries of Jack's body, the only solution is to implement via points manually along the desired pathway.

4.3.11 Executing the Sign

The user interface allows a person to type sentences to be translated. Each word in the sentence is checked against the available words in the dictionary similar to [23]. If the word is found in the dictionary, the file with that sign's parameter definitions is executed, storing these values in the current workspace. If the word is not found to be in the dictionary, the word is fingerspelled. Key location and offset are the same for all letters and numbers, but four different orientations are needed to fully describe the fingerspelled letters and numbers. Four files contain the data for these four situations. Four lists contain the letters associated with each orientation and are consulted before executing one of the four files. For letters G and H, palm orientation is in, pointing horizontally across body. For letters P and Q palm orientation is down, pointing down and away from Jack's body. For numbers 1-5, palm orientation is facing Jack's body, fingers pointing up. For all other numbers and letters, palm faces away from jack, fingers pointing up.

All signs are defined by a finite number of endpoints that are reached for in sequence. The location information for the dominant and non-dominant hands represent the first site reached for, and the location after the movements like 'up', 'down', 'right', 'left' etc. is stored as the second set of locations. If the segment does not involve a designated linear movement the duration time given for the second set of locations is zero. Each site is calculated by combining the translation from the location and offset values with the orientation angles. From segment to segment, Jack's hands move in a linear fashion to a new destination.

Physically moving a reference cube around the scene is useful for trouble shooting and fine tuning the procedure, but computing the location and orientation

mathematically using the vectors and translational matrices uses less computer time and results in a more efficient process.

The beginning and ending location matrices are stored for each segment and are required to define the path. The final transformation matrix is formed by multiplying the orientation array by the translation array that was calculated with the key location and offset information. After all the matrices are calculated, Jack reaches for each in sequence. Hand shape changes and location /orientation changes of the dominant hand are done together in parallel with the nondominant hand. Each segment is completed in sequence. The Reach command makes available the option of designating a time for completion of each step. When the interval is too long, the sign pauses at each endpoint before moving on. For time too short, Jack can not properly align resulting in a choppy or unrealistic motion. Using a PC with higher processing speed sets a quicker signing motion.

This is an improvement on previous methods because it allows each destination to be calculated and stored as needed to describe each specific sign. Only the destinations for one sign are stored at any given time, which reduces the memory needed to execute the program. In previous methods, a cube must already exist, or be created new by the programmer, for a user to use it as a final destination. This creates an ongoing limitation, or reliance on a skilled programmer to expand the signing program. The method of calculating the locations for each sign allows an infinite number of locations to be specified. Even if there are limitations associated with the GUI (i.e. 45 degree intervals associated with orientation), these limits can be overridden by directly editing the text file

created by the GUI. This would not be recommended, but is possible for someone familiar with the variable names and their significance.

4.3.12 Special Features

Several functions have been written to assist the user:

JILL Clears the current workspace and replaces Jack with a female signer.

JACK Clears the current workspace and replaces Jill with a male signer.

DICTIONARY Lists all words available from the sign database.

SPEED User can enter a speed from 1-10. These change the duration in each Reach action in a range from .25-2.50 seconds. One corresponds to 2.5 seconds, the slowest time and 10 corresponds to .25 seconds, the quickest. The default time is 1.5 seconds, level 6. The real time not only corresponds to the time taken for each Reach action, but also the capabilities of the PC running the program.

QUIT Erases all figures in environment and exits from the signing program.

4.4 Recording Parameters

A graphical user interface (GUI) facilitates construction of individual signs to add to the synthesis inventory [see Figure 4.2]. This is the optimum medium for defining the parameters of the sign because it is very user friendly, versatile and easily expandable. Knowing python code or MATLAB code is not necessary to operate the GUI and add to the inventory of signs. For someone familiar with American Sign Language, it is possible to create a sign with a small amount of instruction. Testing for accuracy can be done almost instantly allowing parameters to be immediately changed and retested

without exiting the program. This GUI is implemented in this GUI is contained in Appendix A. This could be downloaded from the following file of the same name. The fig file contains the layout of the GUI and the parameters and constraints:

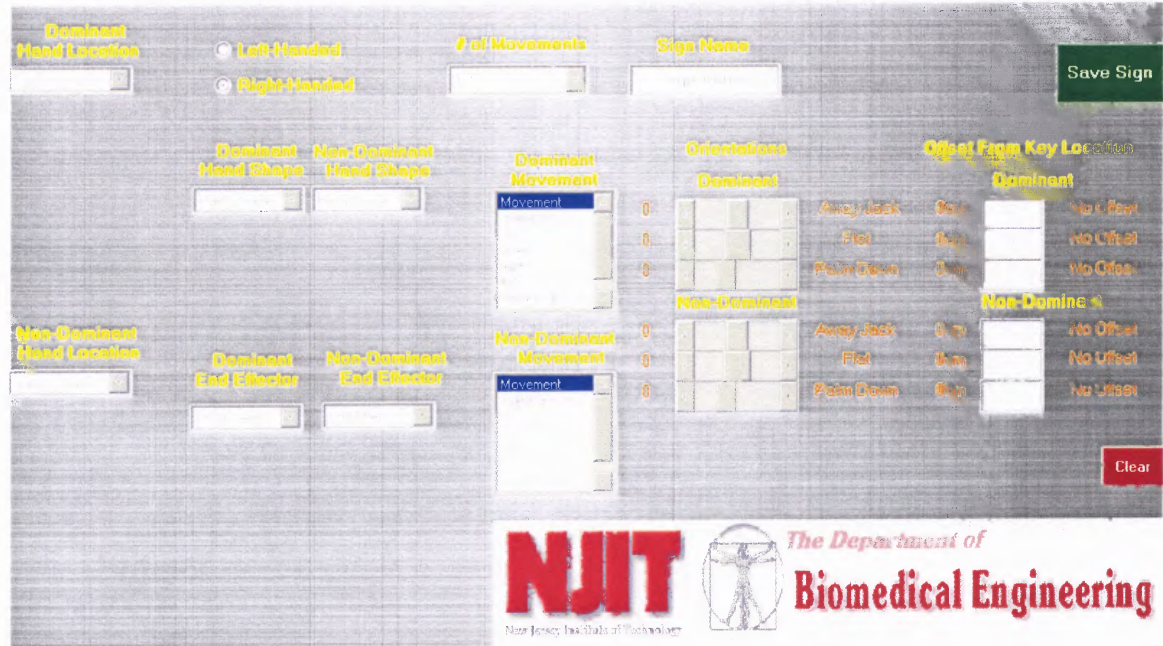


Figure 4.2 Graphical User Interface for parameter setting

4.4.1 Handedness

The option of handedness is a fundamental one, as it is the basis for right or left handedness. Right is the default and dominant preference for signers of ASL. The button is there to demonstrate the ability of the GUI to respond and react to active in this project.

4.4.2 Hand Shape

There are two pull down menus for signers, the hand shape for the dominant and non-dominant hand. When a hand shape is chosen, a picture of the hand shape is displayed under the pull down menu so the user can be sure what the hand shape will look like.

4.4.3 Hand Location and Offset

Location is implemented in a two part fashion, first, the user chooses a body feature from the "Hand Location" pull down menu, then can enter an offset from that location in any or all locations. The user can also choose the inherited location from the previous segment. This is convenient for signs primarily defined by movement sequences. The values follow the coordinates in Jack's environment where up, left and away from Jack are positive offset values and down, right and toward Jack are negative offset values. The default offset is 0 until changed by the user. The offset feature is especially useful for signs that are created away from the body in the "neutral space" in front of the signer's body. The offset is in centimeters and can let the user specify how far away from the body the hand should be located.

4.4.4 Movement

The user is allowed to select more than one movement action by using shift or control while clicking. The effect of actions are combined into one movement, so choosing right and left in the same session will result in no net movement. Similarly, if stationary is selected with another movement, the stationary command is disregarded and has no impact on the movement.

4.4.5 Orientation

Orientation information can be edited using slide bars; text descriptions of the orientation are supplied next to the slide bar. The angles are available in multiples of 45 degrees. The primary reason for this is to make it clear to the user the implications of their choices by giving the most accurate, concise, written descriptions. Orientation information is

implemented in a sequence that is most intuitive for engineers and non-engineers alike. The initial position of the hand is extended in front of Jack, pointing normal to and away from the chest, palm down. Using this as a reference to visualize the rotations needed to achieve the final orientation is important. Rotating first in the axial plane (yaw) keeps palms down while setting up the pointing direction. The pointing direction is defined by the vector starting at the proximal end of the second metacarpal extending through the distal head. Rotating the hand up or down (pitch) results in completely defining the pointing direction. The pointing direction is completely defined by the yaw and pitch parameters; rotating the forearm (roll) results in the palm facing any direction that is within the flexibility of a human.

4.4.6 Multi - Segment Signs

A slide index appears when the user has indicated they would like to use more than one segment to describe their sign. The user can select up to five segments to use in a pull down menu titled '# of segments.' The slide index is hidden when only one segment will be used in order to avoid confusion. The slide index informs the user which segment they are currently editing. When the user adjusts the slide index, if the user has already entered data, they will be able see it in the GUI objects, else, the defaults appear. Only data in segments up to the maximum are recorded. If the user overestimates the amount of segments they will need, then reduces the number later, any edited options chosen above the maximum will not be recorded in the file.

4.4.7 Saving and Clearing Data

The user has two options when finished inputting the data: save or clear. The GUI outputs to a file with a '.py' extension that is generally used in programming python code. The file contains all the parameters just chosen in syntax compatible with python code. For signs represented by one word, the file containing the data will have the same name. Many times signs in ASL cannot be described by simply one word in English; phrases or compound word definitions will be modified using underscores to replace spaces when naming the file.

CHAPTER 5

CONCLUSION

5.1 Discussion

The sign synthesis engine was built with the focus on giving more structure to the function sequence and reducing extraneous objects from Jack's scene. A method of identifying end locations mathematically through implementation of parameters is an important achievement because it makes it unnecessary to create new virtual cubes at every potential end location. In previous methods, an infinite number of cubes could be required because of the need for a cube expressing every potential orientation and location combination. The need to add these cubes also created the need for a trained programmer to be present at the addition of every sign. Keeping track of the names of all these cubes would require an extremely organized labeling system that a new programmer could easily understand.

This method features a sign engine where data for the signs is external to the core of the program. The actual action of the sign is produced with commands containing variable holders for the current sign allowing new features to be implemented without rewriting every sign. A default state can be used for older sign files not containing the variable associated with new features.

The movement feature demonstrates that it is possible to define a movement path between two points and move Jack's hands along the path. It sets the foundation for implementing circular or arced paths. Many signs in American Sign Language make use of circular motion and it is an important feature to offer in a translation program. The signs 'PEOPLE' and 'SIGNING' both involve circular motions. The other group of

movements that must be implemented for more accurate signing is repeated adjustments of the joints in the wrist and hand (i.e. knocking, twisting, flicking, shaking, wiggling etc.).

The end effector at the base of the index finger allows the hand to be oriented and moved to any location regardless of hand shape. Some signs are more accurately described as one figure tip touching another part of the body. This could be one hand in contact with the face, chest or arms, but could also be the left and right hands in contact with each other. Some signs are only defined by a relative location in front of the body with the more significant relationship being the hand interaction. These options are all available by choosing an appropriate end effector and location on the hand.

This method translates the user input word for word with disregard for ASL grammar. American Sign Language has a very complicated grammar structure which can change a sign depending on the context. While singular letters do not change in orientation, the numbers can change appearance depending on the situation they are describing. In general the numbers 1-5 are executed with the palm facing in when referring to quantity. Numbers referring to age, time, addresses, or components of double digit numbers are executed with the palm out.

Although ASL is not a written language, there are attempts to describe the sequence of signs using English equivalents. It is not uncommon in textbooks for signs to be described in this manner; it can be useful and descriptive to those learning ASL as a second language. A translator testing for these context clues and attempting to translate from traditional English to this English adaptation of ASL has yet to be developed and is a weighty linguistical undertaking. Incorporating this theoretical automated English to

ASL grammar translator is an important step in creating an understandable synthetic signer.

In addition to sign sequence, breaking down individual words can also have implications in sign language. Signs depicting professions [see Figure 2.1], often feature a root word and an 'ER' ending. The 'ER' ending which reflects a profession has a sign associated with it. 'Student' is a noted exception, which raises awareness to the complex nature of an ASL to English translator. A sophisticated linguistical analyzer is needed find words with a root-er relationship and to distinguish the difference between 'ER' indicating a profession, and the 'ER' indicating a superlative, as in 'FASTER' and 'TALLER.' Suffixes such as 'ER' and 'EST' indicating a superlative are just two types of suffixes that modify signs in their own individual ways.

The Graphical User Interface is a very important tool allowing a user unfamiliar with the software of the program to create a new sign and test it for accuracy. The GUI attempts to give text indicators for what the user is choosing with regard to angle measurements and movement directions. One small drawback is that once the user clears the sign from the GUI, they must reenter all the data to save the file again. If they are more familiar with the variable names and implementation, they can go directly to the output file without using the GUI and change the options in that way. A mechanism to load a sign file into the GUI would be very beneficial, especially to create similar signs that differ in only a few parameters. A person could load a sign, and change hand shape, for example, and resave as a new sign. The GUI in this method is programmed using MATLAB 6.5. Creating a GUI using a language implemented in Jack, like Python or TCL code will allow the user to interface with Jack directly and see the results of the

choices as they are made. This potential would eliminate the need for trial and error technique and each stage of the sign can be viewed without actually saving the file.

A methodology for evaluation of the quality and accuracy of the signs animated is necessary to justify continuing with the project. If low quality results are being produced, the cause must be sought out and improved. Jack is equipped with a motion capture feature (MoCap). This facilitates use of CyberGloves and Flock of Birds to manipulate Jack's extremities. If a proficient signer uses this equipment to sign, data may be collected to compare with the animation of signs produced synthetically. It is also possible to synchronize the synthetic and captured signing to be performed by two humanoid figures in the same scene in the Jack software. Two possible entities could be the cause for potential deficiencies in this project: human programming error, and limitations of the Jack software.

The Jack Software was designed for ergonomics applications, thus isn't designed for detailed animation of upper extremity movements. Jack's performance in sign language would be improved if there was a way to specify which joint should be the first to adjust for extension to an endpoint. Jack allows reaching from the waist and shoulder. If Jack allowed reaching from the elbow and wrist, signs would reflect energy conservation of normal human movement. If a person can transition from one sign to another adjusting only their forearm, they will save the energy associated with moving their entire limb. Jack is generally void of emotion and there are no joints associated with movement of the mouth or jaw. Facial expressions are important entities in communicating in sign language. The last big improvement would be improved scheme for inverse kinematics. There is often more than one method to get to a mathematical

solution, but the restrictions of the human body's joints are not fully considered when Jack sets out to a destination. This leads to awkward body positioning or simply Jack never reaching the proposed destination.

5.2 Demonstration

A demonstration of the entire program will show the ease of implementation. The signing programming begins with a picture or written description [see Figure 5.1]. This sign will be described using two segments. The hand shape does not change in the duration of the sign, but the orientation changes significantly when moving and eliminates the possibility of using only a single segmented movement.



<http://www.lifeprint.com/asl101/pages-signs/b/belt.htm>

Figure 5.1 Picture Description for 'Belt'

The left and right hands are mirror perform actions that are symmetric about the sagittal plane. Both hands initialize at their respective hips, about 45 degrees rotated inward on the axial plane and twisted 90 degrees at the wrist so that both palm faces are toward the body [See Figure 5.2]. The end position is both hands rotated 90 degrees in

the axial plane, facing across the body, twisted 90 degrees so the palms face the body.

Both hand shapes represent the 'U' fingerspelling hand shape [See Figure 5.3].

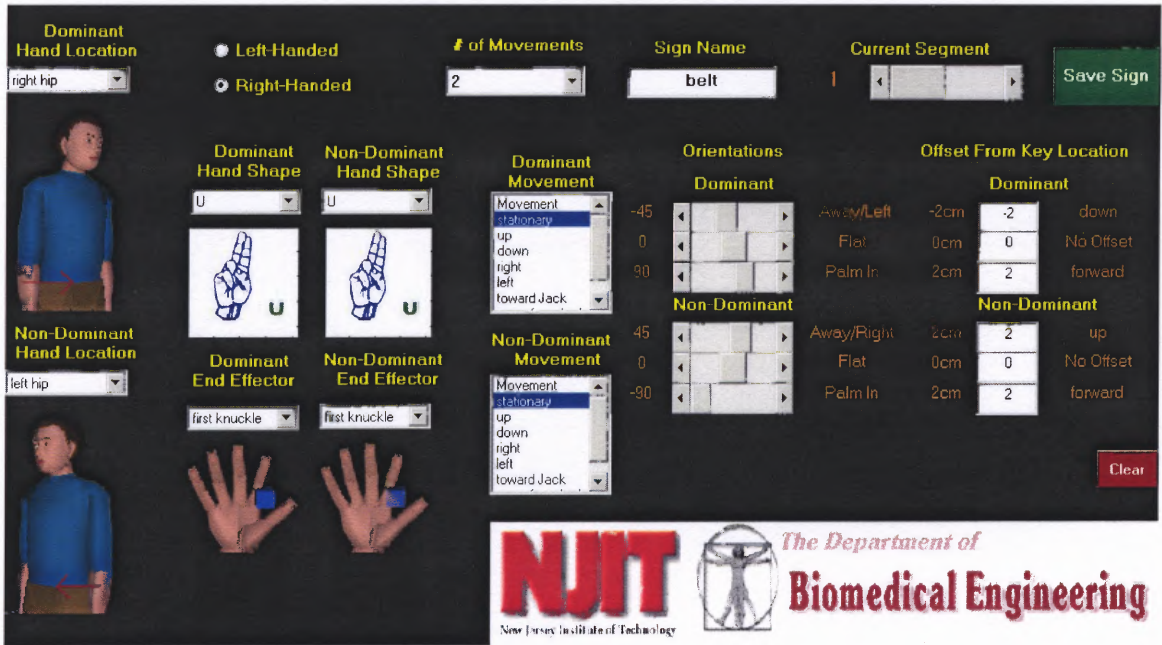


Figure 5.2 Segment One of Sign 'Belt'

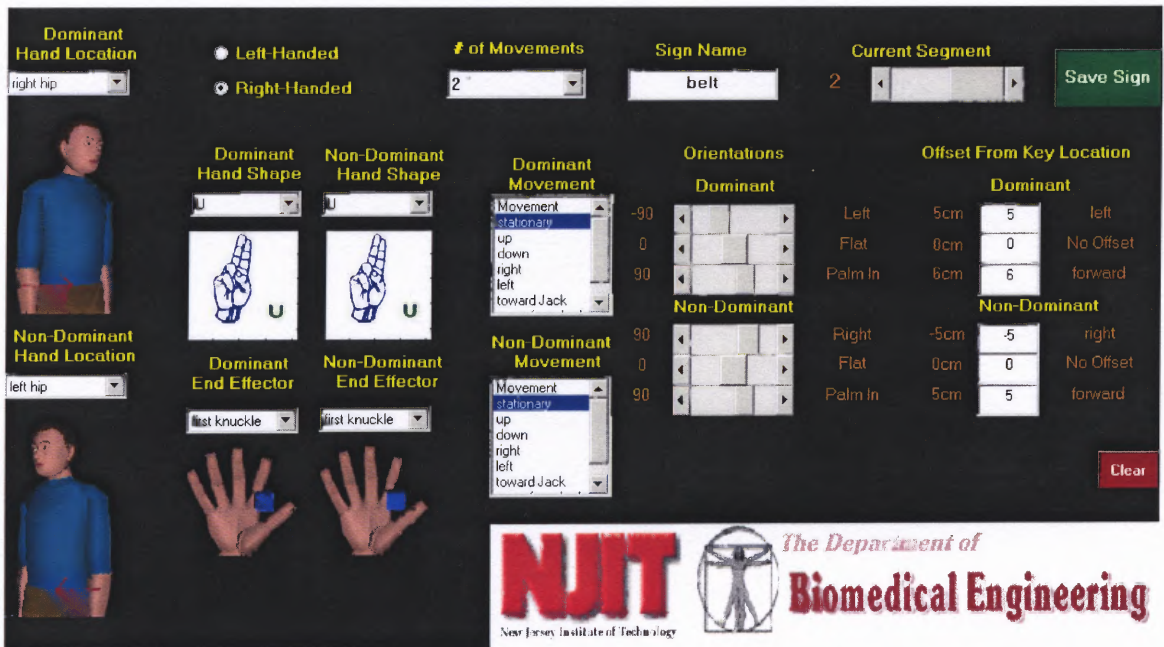


Figure 5.3 Segment Two of Sign 'Belt'

Screen capture of Jack's movement provides evidence for the quality of Jack's signs. Without real time evaluation, motion capture is the next best tool. The results of the sign are depicted in Figure 5.4.

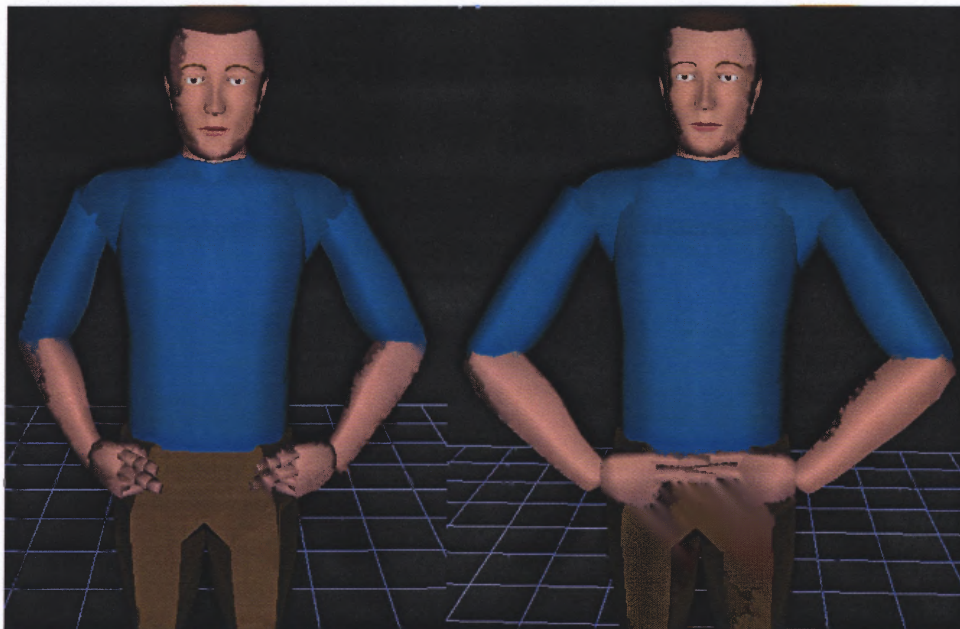


Figure 5.4 Jack Performing 'Belt'

5.3 Breadth of Functionality

The Gallaudet Functional Dictionary, which contains approximately 600 signs for practical every day use, can be used as a benchmark for the progress made in this project, and its future goals. Approximately one third of the signs in this dictionary can be produced immediately using the parameters and tools presented in this document. Another third are possible, but could be produced with more ease and accuracy given more options. Most of this group of signs would be more accurate given an increase in the number of hand locations. The base of the second phalange on the middle finger, as well as the medial and lateral sides of the hand are all good candidates for additions to the

program. The final third could be signed if there were a mechanism for circular and wiggling movements in addition to the linear movements presently allowed. Both vertical and horizontal circular movements are required to create all signs in this dictionary.

5.4 Long Term Vision

Much of this project is motivated by improving the quality of communication between the hearing and the deaf.

5.4.1 Applications for the Hearing

Self-learning American Sign Language is extremely difficult. Static signs like letters and some numbers can be imitated successfully using a sign dictionary or crib sheet, but when the sign involves movement or change in hand shape, portraits leave much to be desired. ASL Dictionary shows real human images to portray the signs, but these can be confusing, misleading, or inaccurate [24]. Video in American Sign Language Browser offers an opportunity to view the movement, but often the camera angle does not change and the details of the hand shape and other subtleties are lost in the third dimension [25].

Imagine a three dimensional virtual teacher to walk around and view through virtual reality glasses from all angles as he is signing. Use virtual reality gloves to manipulate another human in the environment would allow easy comparison between both real and virtual signer.

5.4.2 Applications World Wide

The Jack sign synthesis program is built on such general parameters that it could be applied to any sign language that uses hand shape, position, location, and movement as primary parameters. Hand shapes are not standardized world wide, but additional hand shapes can be programmed using Jack's hand shape tool, which would allow hand shapes uncommon in ASL to become available.

Many sign languages worldwide have a manual alphabet, which allow users to spell out words in the regional written language. Seeing the fingerspelling chart for other sign languages is an indication of the potential success Jack could have in signing languages other than ASL. The Norwegian and German finger alphabets differ from that of ASL only in the letter 'T'. With that, it would seem obvious that Jack could most successfully be programmed to sign in these two languages. The Russian alphabet seems very different from the ASL manual alphabet at first, but upon second look, almost two thirds of the hand shapes are already implemented in the inventory for use with Jack in this project. The British and Australian manual alphabets are both two handed and very similar because Auslan (Australian Sign Language) was inspired by a population of British deaf immigrating to Australia. While a two handed manual alphabet puts more weight on the intricate coordination between the left and right hand, many of the individual hand shapes used in the British alphabet are already used in ASL.

Jack has the potential to be used in conjunction with translation software to animate the written form of a non-English language to the Signed English equivalent. The deaf community has had some success in coming together as an international body, but sometimes communication is reduced to choosing a written language and asking

everyone to memorize the same (often unfamiliar) fingerspelling alphabet. Fingerspelling every word can become tedious, especially when it is not the signer's native alphabet, it can also be difficult to understand in real time. Using Jack's capabilities, it could be possible for a transcriber to describe what the person is saying in text and have this message display in other sign languages on a monitor.

5.4.3 Implications in the Field of Sign Language to English Translation

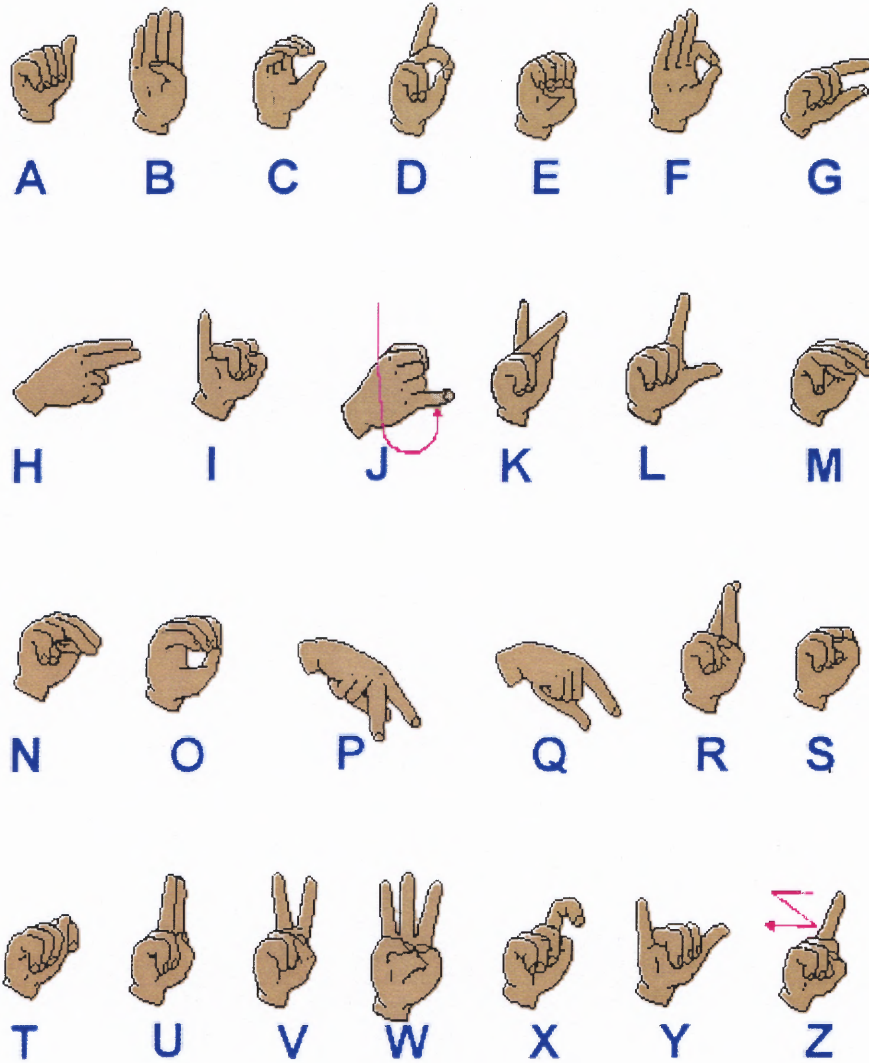
Demonstrating that signs can be synthesized using only parametric data raises the question of whether signs can be translated back into English using the same parametric data. The accepted method for detecting sign language is using motion capture gloves to create a database for all the joint angles, velocities and accelerations plus three-dimensional location information. Gillian Sherry investigated fingerspelling detection using this technique. This technique requires extensive human resources as well as the physical resources required to store the data. The person signing must wear the gloves and the kinematic information is compared with that of the database using a software package capable of statistical analysis and displaying translation into English [26].

A translation technique using parametric data still involves the use of motion capture gloves. Instead of comparing the user with an extensive database of kinematic information, each parameter will be tested and labeled, then compared with the parametric values of each sign. While each sign will still have to exist in the parametric database, this technique can give weights designating the importance of each parameter resulting in a more adaptable model.

APPENDIX A

WORLD FINGERSPELLING

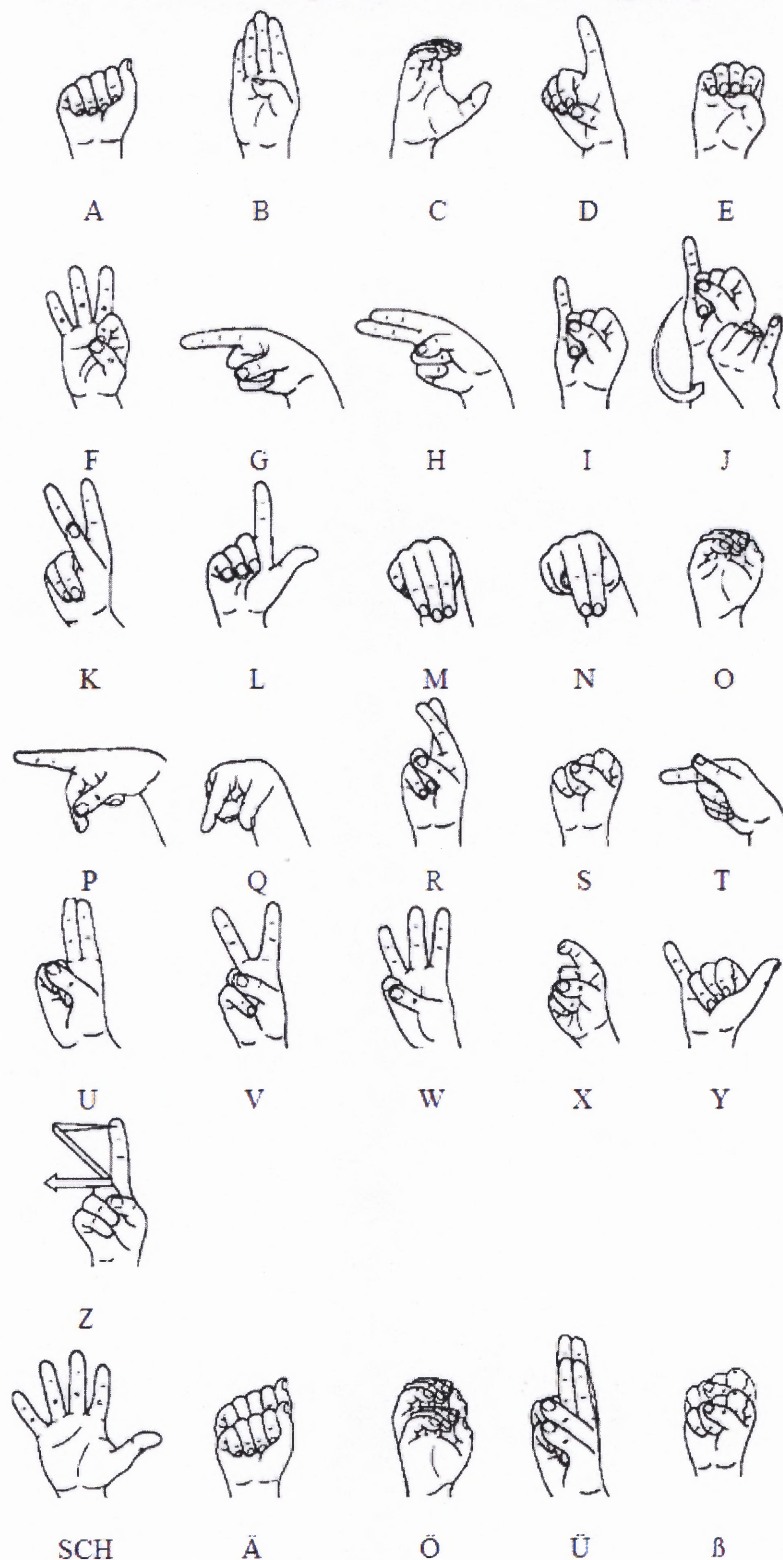
A.1 American Sign Language Fingerspelling Alphabet



<http://www.iidc.indiana.edu/cedir/kidsweb/amachart.html>

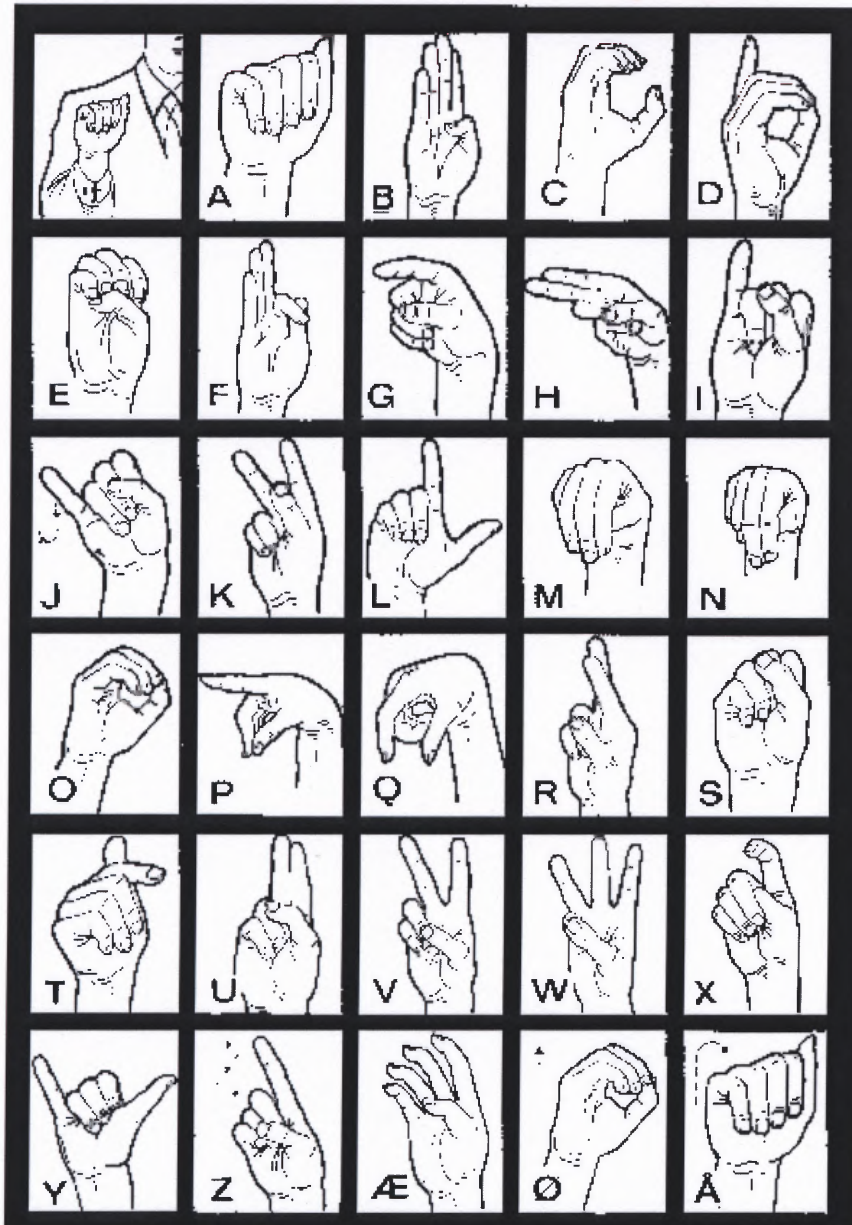
A.2 German Sign Language Fingerspelling Alphabet

German Manual Alphabet / Deutsches Fingeralphabet



A.3 Norwegian Sign Language Fingerspelling Alphabet

The Norwegian Manual Alphabet.





























<http://www.deafblind.com/norwayma.html>

A.4 Russian Sign Language Fingerspelling Alphabet

Russian Manual Alphabet - ДАКТИЛОЛОГИЯ

 А а <i>ah</i> a in art	 Б б <i>beh</i> b in boat	 В в <i>veh</i> v in vote	 Г г <i>geh</i> g in goat	 Д д <i>deh</i> d in deer	 Е е <i>yeh</i> ye in yes
 Ё ё <i>yo</i> yo in your	 Ж ж <i>zheh</i> z in leisure	 З з <i>zeh</i> z in Zen	 И и <i>ee</i> ee in eel	 Й й <i>ee kratkoe</i> y in toy	 К к <i>kah</i> k in kiwi
 Л л <i>el</i> l in elk	 М м <i>em</i> m in mom	 Н н <i>en</i> n in no	 О о <i>oh</i> o in or	 П п <i>peh</i> p in speak	 Р р <i>err</i> trilled r
 С с <i>es</i> s in silk	 Т т <i>teh</i> t in star	 У у <i>oo</i> oo in loon	 Ф ф <i>ef</i> f in fox	 Х х <i>kha</i> ch in Bach	 Ц ц <i>tseh</i> ts in bats
 Ч ч <i>ch eh</i> ch in china	 Ш ш <i>shah</i> sh in shaft	 Щ щ <i>steha</i> shch in fresh cheese	 Ъ ъ tyordyj znak*	 Ь ь <i>yerEE</i> f in ill-no Eng equivalent	 Ъ ъ myagkij znak*
 Э э <i>eh</i> e in Erin	 Ю ю <i>yoo</i> yu in used	 Я я <i>yah</i> ya in yard	* Indicates softness of preceding consonant		

A.5 British Sign Language Fingerspelling Alphabet

BRITISH TWO - HANDED FINGER SPELLING ALPHABET		A 	B 
C 	D 	E 	F 
G 	H 	I 	J 
K 	L 	M 	N 
O 	P 	Q 	R 
S 	T 	U 	V 
W 	X 	Y 	Z 

Source: DeafSign.Com
Date Published @ DS: 14/12/2000

APPENDIX B

MATLAB CODE FOR GRAPHICAL USER INTERFACE

```
% GUI program

% MATLAB code

% allows user to input sign data then prints to file

function varargout = testgui11(varargin)

% Begin initialization code - DO NOT EDIT

gui_Singleton = 1;

gui_State = struct('gui_Name',    mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @untitled_OpeningFcn, ...
    'gui_OutputFcn', @untitled_OutputFcn, ...
    'gui_LayoutFcn', [] , ...
    'gui_Callback', []);

if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```



```

% End initialization code - DO NOT EDIT

% --- Executes just before testgui11 is made visible.

function untitled_OpeningFcn(h, eventdata, handles, varargin)

evalin('base','clear')

handles.output = h;

guidata(h, handles);

global data handed nondom number textlib entries h6 h5 list1 list2 list3 list4 current

current=1;

handed = 'right';

nondom = 'left';

number = 1;

entries = 5;

data=struct('DomShape',{' ',' ',' ',' '},'NonShape',{' ',' ',' ',' '},...
'DomLocale',{' ',' ',' ',' '},'NonLocale',{' ',' ',' ',' '},...
'domval',{1 1 1 1 1},'nonval',{1 1 1 1 1},...
'DomSet',{' ',' ',' ',' '},'NonSet',{' ',' ',' ',' '},...
'DomX',{0 0 0 0 0},'DomY',{0 0 0 0 0},'DomZ',{0 0 0 0 0},...
'NonX',{0 0 0 0 0},'NonY',{0 0 0 0 0},'NonZ',{0 0 0 0 0},...
'DomOffX',{0 0 0 0 0},'DomOffY',{0 0 0 0 0},...
'DomOffZ',{0 0 0 0 0},'NonOffX',{0 0 0 0 0},...
'NonOffY',{0 0 0 0 0},'NonOffZ',{0 0 0 0 0},...
'DomEffector',{' ',' ',' ',' '},'NonEffector',{' ',' ',' ',' '});

textlib=struct('a',{'Toward Jack','Toward/Left','Left','Away/Left',...

```

```

    'Away Jack','Away/Right','Right','Toward/Right','Toward Jack'},...
    'b',{ 'Palm Up','Palm In/Up','Palm In','Palm In/Down','Palm Down',...
        'Palm Out/Down','Palm Out','"},...
    'c',{ 'Up','Flat/Up','Flat','Flat/Down','Down',' ',' ',' '});
% h5 and h6 are struct of handles
h6=struct('text1823',{ handles.text18,handles.text19,handles.text20,...
    handles.text21,handles.text22,handles.text23 },...
'text2429',{ handles.text24,handles.text25,handles.text26,...
    handles.text27,handles.text28,handles.text29 },...
'rotslide',{ handles.DomSlideX,handles.DomSlideY,handles.DomSlideZ,...
    handles.NonSlideX,handles.NonSlideY,handles.NonSlideZ },...
'text3540',{ handles.text35,handles.text36,handles.text37,...
    handles.text38,handles.text39,handles.text40 },...
'text4146',{ handles.text41,handles.text42,handles.text43,...
    handles.text44,handles.text45,handles.text46 },...
'offset',{ handles.DomOffX,handles.DomOffY,handles.DomOffZ,...
    handles.NonOffX,handles.NonOffY,handles.NonOffZ });
% hand shape options offered to user
list1={'Hand Shape','rest','A','B','C','D','E','F','G','H','I','J',...
    'K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y',...
    'Z','1','2','3','4','5','6','7','8','9','10','OpenB','Plane',...
    'BentB','BentV','ClawShape','FlatO','reset','neutral'};
% hand location options offered to user

```

```

list2={'Hand Location','inherited','left brow','forehead',...
      'right brow','nose','left ear','right ear','chin','neck',...
      'left shoulder','shoulder center','right shoulder',...
      'left elbow','right elbow','chest left','chest center','chest right',...
      'left hip','hip center','right hip','left wrist','right wrist',...
      'right first knuckle','left first knuckle','right index finger',...
      'left index finger','right middle finger','left middle finger',...
      'right ring finger','left ring finger','right pinky finger',...
      'left pinky finger','right thumb','left thumb',...
      'right palm center','left palm center'};

% movement selections offered to user

list3={'Movement','stationary','up','down','right','left',...
      'toward Jack','away from Jack'};

list4={'End Effector','first knuckle','index finger','middle finger',...
      'ring finger','pinky finger','thumb','palm center'};

set(handles.DomSlideY,'Max',4)
set(handles.DomSlideY,'Min',-2)
set(handles.NonSlideY,'Max',4)
set(handles.NonSlideY,'Min',-2)

% Re: display a njitlogo by Ana

% Ana <ana_mike2000@yahoo.com>, 24 Apr 03 10:35:34 -0400 (EDT)

% http://mathforum.org/epigone/comp.soft-
sys.matlab/zimpfumsmen/e2jgimcwwkeg@legacy

```

```
Graphic = imread('picture/bmenjit.bmp', 'bmp');  
gInfo = imfinfo('picture/bmenjit.bmp', 'bmp');  
position = get(handles.njitlogo, 'Position');  
position(3) = gInfo.Width;  
position(4) = gInfo.Height;  
axes(handles.njitlogo);  
image(Graphic);  
  
% Fix the axis so that X and Y are not visible  
set(handles.njitlogo,'XTick',[])  
set(handles.njitlogo,'YTick',[])  
  
set(handles.DomHandShape,'String',list1)  
set(handles.NonHandShape,'String',list1)  
set(handles.DomHandLocale,'String',list2)  
set(handles.NonHandLocale,'String',list2)  
set(handles.DomMoveSet,'String',list3)  
set(handles.NonMoveSet,'String',list3)  
set(handles.DomEndEffector,'String',list4)  
set(handles.NonEndEffector,'String',list4)  
  
temp = {'Away Jack','Palm Down','Flat','Away Jack',...  
        'Palm Down','Flat'};  
  
for r = 1:6
```

```

set(h6(r).text1823,'String',0)

set(h6(r).text2429,'String',temp(r))

set(h6(r).rotslide,'Value',0)

set(h6(r).text3540,'String','0cm')

set(h6(r).text4146,'String','No Offset')

set(h6(r).offset,'String','0')

end

% --- Executes on button press in cleargui.

function cleargui_Callback(h, eventdata, handles)

evalin('base','clear')

global data handed number textlib entries h6 h5

handed = 'right';

nondom = 'left';

number = 1;

entries = 5;

data=struct('DomShape',{' ',' ',' ',' '},'NonShape',{' ',' ',' ',' '},...

'DomLocale',{' ',' ',' ',' '},'NonLocale',{' ',' ',' ',' '},...

'domval',{1 1 1 1 1},'nonval',{1 1 1 1 1},...

'DomSet',{' ',' ',' ',' '},'NonSet',{' ',' ',' ',' '},...

'DomX',{0 0 0 0 0},'DomY',{0 0 0 0 0},'DomZ',{0 0 0 0 0},...

'NonX',{0 0 0 0 0},'NonY',{0 0 0 0 0},'NonZ',{0 0 0 0 0},...

'DomEffector',{' ',' ',' ',' '},'NonEffector',{' ',' ',' ',' '});

```

```

textlib=struct('a',{ 'Toward Jack','Toward/Left','Left','Away/Left',...
    'Away Jack','Away/Right','Right','Toward/Right','Toward Jack'},...
'b',{ 'Palm Up','Palm In/Up','Palm In','Palm In/Down','Palm Down',...
    'Palm Out/Down','Palm Out','"},...
'c',{ 'Up','Flat/Up','Flat','Flat/Down','Down','','','});

```

```

temp = { 'Away Jack','Palm Down','Flat','Away Jack',...
    'Palm Down','Flat'};

```

```

for r=1:6

```

```

    %resets rotation value indicator

```

```

    set(h6(r).text1823,'String',0)

```

```

    %resets offset value indicator

```

```

    set(h6(r).text3540,'String','0cm')

```

```

    %resets rotation direction indicator

```

```

    set(h6(r).text2429,'String',temp(r))

```

```

    %resets offset direction indicator

```

```

    set(h6(r).text4146,'String','No Offset')

```

```

    %resets offset input

```

```

    set(h6(r).offset,'String','')

```

```

    %resets slider values

```

```

    set(h6(r).rotslide,'Value',0)

```

```

end

```

```

% reset hand preference to right (default)

```

```
set(handles.right,'Value',1)

set(handles.left,'Value',0)

% reset number of movements to 1,

% set maximum on slide index and visibility of pull downs

set(handles.NumMove,'Value',1)

set(handles.index,'String',1)

set(handles.slide_index,'Max',number)

set(handles.slide_index,'Visible','off')

set(handles.index,'Visible','off')

set(handles.text14,'Visible','off')

%resets each pull down to first value in String

set(handles.NonMoveSet,'Value',1)

set(handles.DomMoveSet,'Value',1)

set(handles.NonHandLocale,'Value',1)

set(handles.DomHandLocale,'Value',1)

set(handles.NonHandShape,'Value',1)

set(handles.DomHandShape,'Value',1)

set(handles.DomEndEffector,'Value',1)

set(handles.NonEndEffector,'Value',1)

%resets sign name

set(handles.signnameedit,'String','sign name')
```

% --- Outputs from this function are returned to the command line.

```
function varargout = untitled_OutputFcn(h, eventdata, handles)
```

```
varargout{1} = handles.output;
```

```
global handed number sign_name
```

```
%Radio Buttons
```

```
function mutual_exclude(off)
```

```
set(off, 'Value', 0)
```

```
%Pick left hand
```

```
function left_Callback(h, eventdata, handles)
```

```
global h6 handed nondom data textlib entries
```

```
off = [handles.right];
```

```
mutual_exclude(off)
```

```
if strcmp(handed, 'left')
```

```
    set(handles.left, 'Value', 1)
```

```
end
```

```
n=1:entries;
```

```
if strcmp(handed, 'right')
```

```
    % rotation about X and Y axis when switching handedness
```

```
    for n = 1:entries
```

```
        data(n).DomY=-data(n).DomY;
```



```

    data(n).NonY=-data(n).NonY;

    data(n).DomX=-data(n).DomX;

    data(n).NonX=-data(n).NonX;

end

current = round(get(handles.slide_index,'Value'));

% current variable in slide index

T = [data(current).DomX,data(current).DomY,data(current).DomZ,...
     data(current).NonX,data(current).NonY,data(current).NonZ];

temp = struct('temp',[cellstr(textlib(T(1)/45+5).a),...
                    cellstr(textlib(T(2)/45+5).b),cellstr(textlib(T(3)/45+3).c),...
                    cellstr(textlib(T(4)/45+5).a),cellstr(textlib(-T(5)/45+5).b),...
                    cellstr(textlib(T(6)/45+3).c))]);

for r=1:6

    set(h6(r).text1823,'String',T(r))

    set(h6(r).rotslide,'Value',round(T(r)/45))

    set(h6(r).text2429,'String',temp(r).temp)

end

end

handed = 'left';

nondom = 'right';

%Pick right hand

function right_Callback(h, eventdata, handles)

```

```

global handed nondom data textlib entries h6

off = [handles.left];

mutual_exclude(off)

if strcmp(handed, 'right')
    set(handles.right,'Value',1)
end

if strcmp(handed, 'left')
    % rotation about X and Y axis when switching handedness

    for n = 1: entries
        data(n).DomY=-data(n).DomY;
        data(n).NonY=-data(n).NonY;
        data(n).DomX=-data(n).DomX;
        data(n).NonX=-data(n).NonX;
    end

    current = round(get(handles.slide_index,'Value'));

    % current variable in slide index

    T = [data(current).DomX,data(current).DomY,data(current).DomZ,...
        data(current).NonX,data(current).NonY,data(current).NonZ];

    temp = struct('temp',[cellstr(textlib(T(1)/45+5).a),...
        cellstr(textlib(-T(2)/45+5).b),cellstr(textlib(T(3)/45+3).c),...
        cellstr(textlib(T(4)/45+5).a),cellstr(textlib(T(5)/45+5).b),...
        cellstr(textlib(T(6)/45+3).c)]);

    for r=1:6

```

```

    set(h6(r).text1823,'String',T(r))

    set(h6(r).rotslide,'Value',round(T(r)/45))

    set(h6(r).text2429,'String',temp(r).temp)

    end

end

handed = 'right';

nondom = 'left';

% Number of Movements

function NumMove_CreateFcn(h, eventdata, handles)

if ispc

    set(h,'BackgroundColor','white');

else

    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

function NumMove_Callback(h, eventdata, handles)

global number h5

number=get(h,'Value');

if number ~= 1

    set(handles.slide_index,'SliderStep',[1/(number-1),1/(number-1)])

end

if number == 1

    set(handles.index,'String',1)

```

```
set(handles.slide_index,'Max',number)

set(handles.slide_index,'Visible','off')

set(handles.index,'Visible','off')

set(handles.text14,'Visible','off')

temp = {'on','off','off','off','off'};

else

set(handles.slide_index,'Max',number)

set(handles.slide_index,'Visible','on')

set(handles.index,'Visible','on')

set(handles.text14,'Visible','on')

end

% Sign name

function signnameedit_CreateFcn(h, eventdata, handles)

global sign_name

if ispc

set(h,'BackgroundColor','white');

else

set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

function signnameedit_Callback(h, eventdata, handles)

global sign_name

sign_name = get(h,'string');
```

```

% Slide Index

function slide_index_CreateFcn(h, eventdata, handles)

usewhitebg = 1;

if usewhitebg

    set(h,'BackgroundColor',[.9 .9 .9]);

else

    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

function slide_index_Callback(h, eventdata, handles)

global current data entries textlib handed h6 h5 list1 list2 list3 list4

current = get(handles.slide_index,'Value');

current = round(current);

set(handles.index,'String',current)

% Sets popup menu to reflect previously chosen value

h=1;

if strcmp(data(current).DomShape,"")==0

    while strcmp(data(current).DomShape,list1(h))==0

        h=h+1;

    end

end

set(handles.DomHandShape,'Value',h)

```

```
h=1;

if strcmp(data(current).NonShape,"")==0

    while strcmp(data(current).NonShape,list1(h))==0

        h=h+1;

    end

end

set(handles.NonHandShape,'Value',h)

h=1;

if strcmp(data(current).DomLocale,"")==0

    while strcmp(data(current).DomLocale,list2(h))==0

        h=h+1;

    end

end

set(handles.DomHandLocale,'Value',h)

h=1;

if strcmp(data(current).NonLocale,"")==0

    while strcmp(data(current).NonLocale,list2(h))==0

        h=h+1;

    end

end

set(handles.NonHandLocale,'Value',h)

h=1;
```

```

if strcmp(data(current).DomEffector,"")==0
    while strcmp(data(current).DomEffector,list4(h))==0
        h=h+1;
    end
end

set(handles.DomEndEffector,'Value',h)

h=1;
if strcmp(data(current).NonEffector,"")==0
    while strcmp(data(current).NonEffector,list4(h))==0
        h=h+1;
    end
end

set(handles.NonEndEffector,'Value',h)

set(handles.DomMoveSet,'Value',data(current).domval)
set(handles.NonMoveSet,'Value',data(current).nonval)

T = [data(current).DomX,data(current).DomY,data(current).DomZ,...
    data(current).NonX,data(current).NonY,data(current).NonZ,...
    data(current).DomOffX,data(current).DomOffY,...
    data(current).DomOffZ,data(current).NonOffX,...
    data(current).NonOffY,data(current).NonOffZ];

if strcmp(handed, 'right')

```

```

temp = struct('temp',[cellstr(textlib(T(1)/45+5).a),...
    cellstr(textlib(-T(2)/45+5).b),cellstr(textlib(T(3)/45+3).c),...
    cellstr(textlib(T(4)/45+5).a),cellstr(textlib(T(5)/45+5).b),...
    cellstr(textlib(T(6)/45+3).c)]);

else

temp = struct('temp',[cellstr(textlib(T(1)/45+5).a),...
    cellstr(textlib(T(2)/45+5).b),cellstr(textlib(T(3)/45+3).c),...
    cellstr(textlib(T(4)/45+5).a),cellstr(textlib(-T(5)/45+5).b),...
    cellstr(textlib(T(6)/45+3).c)]);

end

temp1=struct('temp',["","","",""]);

for r=1:6

    set(h6(r).text1823,'String',T(r))

    set(h6(r).rotslide,'Value',round(T(r)/45))

    set(h6(r).text2429,'String',temp(r).temp)

    set(h6(r).text3540,'String',strcat(num2str(T(r+6)),'cm'))

    set(h6(r).offset,'String',T(r+6))

    if str2num(get(h6(r).offset,'String')) == 0

        temp1(r).temp = 'No Offset';

    elseif findstr(r,[1,4])

        if str2num(get(h6(r).offset,'String'))>0

            temp1(r).temp = 'up';

        else

```



```
        temp1(r).temp ='down';
    end
elseif findstr(r,[2,5])
    if str2num(get(h6(r).offset,'String'))>0
        temp1(r).temp = 'left';
    else
        temp1(r).temp ='right';
    end
elseif findstr(r,[3,6])
    if str2num(get(h6(r).offset,'String'))>0
        temp1(r).temp = 'forward';
    else
        temp1(r).temp ='backward';
    end
end
end
set(h6(r).text4146,'String',temp1(r).temp)
end

% Dominant Hand Shape
function DomHandShape_CreateFcn(h, eventdata, handles)
if ispc
    set(h,'BackgroundColor','white');
else
```

```
    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

set(h,'Visible','on')

function DomHandShape_Callback(h, eventdata, handles)

global list1 data current

val=get(h,'Value');

data(current).DomShape=list1(val);

Graphic = imread(char(strcat('handshapes/',list1(val),'.bmp')), 'bmp');

gInfo = imfinfo(char(strcat('handshapes/',list1(val),'.bmp')), 'bmp');

position = get(handles.DHS, 'Position');

position(3) = gInfo.Width;

position(4) = gInfo.Height;

axes(handles.DHS);

image(Graphic);

% Non Dominant Hand Shape

function NonHandShape_CreateFcn(h, eventdata, handles)

if ispc

    set(h,'BackgroundColor','white');

else

    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

set(h,'Visible','on')
```

```

function NonHandShape_Callback(h, eventdata, handles)

global list1 data current

val=get(h,'Value');

data(current).NonShape=list1(val);

Graphic = imread(char(strcat('handshapes/',list1(val),'.bmp')), 'bmp');

gInfo = imfinfo(char(strcat('handshapes/',list1(val),'.bmp')), 'bmp');

position = get(handles.NDHS, 'Position');

position(3) = gInfo.Width;

position(4) = gInfo.Height;

axes(handles.NDHS);

image(Graphic);

% Dominant Hand Location

function DomHandLocale_CreateFcn(h, eventdata, handles)

if ispc

    set(h,'BackgroundColor','white');

else

    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

set(h,'Visible','on')

function DomHandLocale_Callback(h, eventdata, handles)

global data current list2

val=get(h,'Value');

```

```

data(current).DomLocale=list2(val);

Graphic = imread(char(strcat('locations/',list2(val),'.bmp')), 'bmp');

gInfo = imfinfo(char(strcat('locations/',list2(val),'.bmp')), 'bmp');

position = get(handles.DL, 'Position');

position(3) = gInfo.Width;

position(4) = gInfo.Height;

axes(handles.DL);

image(Graphic);

% Non Dominant Hand Location

function NonHandLocale_CreateFcn(h, eventdata, handles)

if ispc

    set(h,'BackgroundColor','white');

else

    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

set(h,'Visible','on')

function NonHandLocale_Callback(h, eventdata, handles)

global data list2 current

val=get(h,'Value');

data(current).NonLocale=list2(val);

Graphic = imread(char(strcat('locations/',list2(val),'.bmp')), 'bmp');

gInfo = imfinfo(char(strcat('locations/',list2(val),'.bmp')), 'bmp');

```

```

position = get(handles.NDL, 'Position');

position(3) = gInfo.Width;

position(4) = gInfo.Height;

axes(handles.NDL);

image(Graphic);

%Selects End Effector for Dominant hand

function DomEndEffector_CreateFcn(h, eventdata, handles)

if ispc

    set(h,'BackgroundColor','white');

else

    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

function DomEndEffector_Callback(h, eventdata, handles)

global list4 data current

val=get(h,'Value');

data(current).DomEffector=list4(val);

Graphic = imread(char(strcat('handshapes/',list4(val),'.bmp')), 'bmp');

gInfo = imfinfo(char(strcat('handshapes/',list4(val),'.bmp')), 'bmp');

position = get(handles.DEE, 'Position');

position(3) = gInfo.Width;

position(4) = gInfo.Height;

axes(handles.DEE);

```

```
image(Graphic);
```

```
%Selects End Effector for Non-Dominant hand
```

```
function NonEndEffector_CreateFcn(h, eventdata, handles)
```

```
if ispc
```

```
    set(h,'BackgroundColor','white');
```

```
else
```

```
    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
```

```
end
```

```
function NonEndEffector_Callback(h, eventdata, handles)
```

```
global list4 data current
```

```
val=get(h,'Value');
```

```
data(current).NonEffector=list4(val);
```

```
Graphic = imread(char(strcat('handshapes/',list4(val),'.bmp')), 'bmp');
```

```
gInfo = imfinfo(char(strcat('handshapes/',list4(val),'.bmp')), 'bmp');
```

```
position = get(handles.NDEE, 'Position');
```

```
position(3) = gInfo.Width;
```

```
position(4) = gInfo.Height;
```

```
axes(handles.NDEE);
```

```
image(Graphic);
```

```
% Dominant Movement Set
```

```
function DomMoveSet_CreateFcn(h, eventdata, handles)
```

```

if ispc
    set(h,'BackgroundColor','white');
else
    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

set(h,'Visible','on')

function DomMoveSet_Callback(h, eventdata, handles)

global data current list3

data(current).domval=get(h,'Value');

data(current).DomSet=char(cellstr(list3(data(current).domval)));

% Non Dominant Movement Set

function NonMoveSet_CreateFcn(h, eventdata, handles)

if ispc
    set(h,'BackgroundColor','white');
else
    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

set(h,'Visible','on')

function NonMoveSet_Callback(h, eventdata, handles)

global data current list3

data(current).nonval=get(h,'Value');

data(current).NonSet=char(cellstr(list3(data(current).nonval)));

```

```

% Create text file

function run_Callback(h, eventdata, handles)

global handed nondom number sign_name data

underscore_sign_name = sign_name;

a = find(sign_name == ' ');

underscore_sign_name(a) = '_';

fid = fopen(strcat(underscore_sign_name, '.py'), 'w');

fprintf(fid, '# %s \n\n', sign_name);

fprintf(fid, 'number = %d\nhanded = "%s"', number, handed);

fprintf(fid, '\nnondom = "%s"\n\nsign_name = "%s"\n', nondom, sign_name);

temp1 = {'DomShape', 'NonShape', 'DomLocale', 'NonLocale', 'DomEffector', ...
        'NonEffector', 'DomX', 'DomY', 'DomZ', 'NonX', 'NonY', 'NonZ', ...
        'DomOffX', 'DomOffY', 'DomOffZ', 'NonOffX', 'NonOffY', 'NonOffZ'};

assignin('base', 'data', data)

for n = 1:number

    temp(n) = struct('temp', { struct('temp', { data(n).DomShape, data(n).NonShape, ...
        data(n).DomLocale, data(n).NonLocale, data(n).DomEffector, ...
        data(n).NonEffector, data(n).DomX, data(n).DomY, ...
        data(n).DomZ, data(n).NonX, data(n).NonY, data(n).NonZ, ...
        data(n).DomOffX, data(n).DomOffY, data(n).DomOffZ, ...
        data(n).NonOffX, data(n).NonOffY, data(n).NonOffZ } )});

end

```



```

assignin('base','temp1',temp1)

for m=1:6

    fprintf(fid,'\n%s = ["%s",...
        char(temp1(m)),char(cellstr(temp(1).temp(:,m).temp))));

    for n=2:number

        fprintf(fid,' "%s"',char(cellstr(temp(n).temp(:,m).temp)));

    end

    fprintf(fid,']');

end

for m=7:18

    fprintf(fid,'\n%s = [%d',...
        char(temp1(m)),temp(1).temp(m).temp);

    for n=2:number

        fprintf(fid,' %d',temp(n).temp(:,m).temp);

    end

    fprintf(fid,']');

end

fprintf(fid,'\nDomSet = []);

for n=1:number

    ff=size(data(n).DomSet);

    f=ff(1);

    if f == 1

```

```

    fprintf(fid,('%s)',char(cellstr(data(n).DomSet(1,:))));
else
    fprintf(fid,('%s"',char(cellstr(data(n).DomSet(1,:))));

    if f > 1
        for b=2:f
            fprintf(fid,','"%s"',char(cellstr(data(n).DomSet(b,:))));
        end
    end

    fprintf(fid,')');

    if n ~= number
        fprintf(fid,',' ');
    end

end

end

end

fprintf(fid,']');

fprintf(fid,'\nNonSet = []);

for n=1:number

    ff=size(data(n).NonSet);

    f=ff(1);

    if f == 1

        fprintf(fid,('%s)',char(cellstr(data(n).NonSet(1,:))));

    else

```

```
fprintf(fid,("%s",char(cellstr(data(n).NonSet(1,:))));  
  
if f >1  
  
for b=2:f  
  
    fprintf(fid,("%s",char(cellstr(data(n).NonSet(b,:))));  
  
end  
  
end  
  
fprintf(fid,');  
  
if n ~= number  
  
    fprintf(fid, ' ');  
  
end  
  
end  
  
end  
  
end  
  
fprintf(fid,']');  
  
fprintf(fid,'\n');  
  
fclose(fid);  
  
  
function DomSlideX_CreateFcn(h, eventdata, handles)  
  
usewhitebg = 1;  
  
if usewhitebg  
  
    set(h,'BackgroundColor',[.9 .9 .9]);  
  
else  
  
    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));  
  
end
```

```
function DomSlideX_Callback(h, eventdata, handles)
```

```
global data textlib
```

```
DomX=45*round(get(h,'Value'));
```

```
I=round(get(handles.slide_index,'Value'));
```

```
set(handles.text18,'String',DomX)
```

```
temp = textlib(DomX/45+5).a;
```

```
set(handles.text24,'String',temp)
```

```
data(I).DomX = DomX;
```

```
function DomSlideY_CreateFcn(h, eventdata, handles)
```

```
usewhitebg = 1;
```

```
if usewhitebg
```

```
    set(h,'BackgroundColor',[.9 .9 .9]);
```

```
else
```

```
    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
```

```
end
```

```
function DomSlideY_Callback(h, eventdata, handles)
```

```
global handed textlib data
```

```
DomY=45*round(get(handles.DomSlideY,'Value'));
```

```
I=round(get(handles.slide_index,'Value'));
```

```
set(handles.text19,'String',DomY)
```

```
if strcmp(handed, 'left')
```

```
temp=textlib(-DomY/45+5).b;

else

temp=textlib(-DomY/45+5).b;

end

set(handles.text25,'String',temp)

data(I).DomY = DomY;

function DomSlideZ_CreateFcn(h, eventdata, handles)

usewhitebg = 1;

if usewhitebg

set(h,'BackgroundColor',[.9 .9 .9]);

else

set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

function DomSlideZ_Callback(h, eventdata, handles)

global data textlib

DomZ=45*round(get(handles.DomSlideZ,'Value'));

I=round(get(handles.slide_index,'Value'));

set(handles.text20,'String',DomZ)

temp=textlib(DomZ/45+3).c;

set(handles.text26,'String',temp)

data(I).DomZ = DomZ;
```

```

function NonSlideX_CreateFcn(h, eventdata, handles)

usewhitebg = 1;

if usewhitebg

    set(h,'BackgroundColor',[.9 .9 .9]);

else

    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

function NonSlideX_Callback(h, eventdata, handles)

global data textlib

NonX=45*round(get(handles.NonSlideX,'Value'));

I=round(get(handles.slide_index,'Value'));

set(handles.text21,'String',NonX)

temp = textlib(NonX/45+5).a;

set(handles.text27,'String',temp)

data(I).NonX = -NonX;

function NonSlideY_CreateFcn(h, eventdata, handles)

usewhitebg = 1;

if usewhitebg

    set(h,'BackgroundColor',[.9 .9 .9]);

else

    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

```

```
end

function NonSlideY_Callback(h, eventdata, handles)

global data textlib handed

NonY=45*round(get(handles.NonSlideY,'Value'));

I=round(get(handles.slide_index,'Value'));

set(handles.text22,'String',NonY)

if strcmp(handed, 'left')

    temp=textlib(-NonY/45+5).b;

else

    temp=textlib(-NonY/45+5).b;

end

set(handles.text28,'String',temp)

data(I).NonY = -NonY;

function NonSlideZ_CreateFcn(h, eventdata, handles)

usewhitebg = 1;

if usewhitebg

    set(h,'BackgroundColor',[.9 .9 .9]);

else

    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

function NonSlideZ_Callback(h, eventdata, handles)

global data textlib
```

```

NonZ=45*round(get(handles.NonSlideZ,'Value'));

I=round(get(handles.slide_index,'Value'));

set(handles.text23,'String',NonZ)

temp=textlib(NonZ/45+3).c;

set(handles.text29,'String',temp)

data(I).NonZ = -NonZ;

% X Offset of Dominant Hand

function DomOffX_CreateFcn(h, eventdata, handles)

if ispc
    set(h,'BackgroundColor','white');
else
    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function DomOffX_Callback(h, eventdata, handles)

global data

tempstring=strcat(get(h,'String'),'cm');

set(handles.text35,'String',tempstring)

if str2num(get(h,'String'))>0
    tempstring = 'left';
elseif str2num(get(h,'String'))==0
    tempstring = 'No Offset';
else

```



```

    tempstring ='right';

end

set(handles.text41,'String',tempstring);

%stores current value of Dominant Offset X

data(get(handles.slide_index,'Value')).DomOffX=...

    str2num(get(handles.DomOffX,'String'));

%Y Offset of Dominant Hand

function DomOffY_CreateFcn(h, eventdata, handles)

if ispc

    set(h,'BackgroundColor','white');

else

    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

function DomOffY_Callback(h, eventdata, handles)

global data

tempstring=strcat(get(h,'String'),'cm');

set(handles.text36,'String',tempstring)

if str2num(get(h,'String'))>0

    tempstring = 'up';

elseif str2num(get(h,'String'))==0

    tempstring = 'No Offset';

else

```

```

    tempstring ='down';

end

set(handles.text42,'String',tempstring);

%stores current value of Dominant Offset Y
data(get(handles.slide_index,'Value')).DomOffY=...
    str2num(get(handles.DomOffY,'String'));

%Z Offset of Dominant Hand
function DomOffZ_CreateFcn(h, eventdata, handles)
if ispc
    set(h,'BackgroundColor','white');
else
    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function DomOffZ_Callback(h, eventdata, handles)

global data

tempstring=strcat(get(h,'String'),'cm');

set(handles.text37,'String',tempstring)

if str2num(get(h,'String'))>0
    tempstring = 'forward';
elseif str2num(get(h,'String'))==0
    tempstring = 'No Offset';
else

```

```

    tempstring ='backward';

end

set(handles.text43,'String',tempstring);

%stores current value of Dominant Offset Z

data(get(handles.slide_index,'Value')).DomOffZ=...

    str2num(get(handles.DomOffZ,'String'));

% X Offset of Non-Dominant Hand

function NonOffX_CreateFcn(h, eventdata, handles)

if ispc

    set(h,'BackgroundColor','white');

else

    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

function NonOffX_Callback(h, eventdata, handles)

global data

tempstring=strcat(get(h,'String'),'cm');

set(handles.text38,'String',tempstring)

if str2num(get(h,'String'))>0

    tempstring = 'left';

elseif str2num(get(h,'String'))==0

    tempstring = 'No Offset';

else

```

```

    tempstring ='right';

end

set(handles.text44,'String',tempstring);

%stores current value of Non-Dominant Offset X
data(get(handles.slide_index,'Value')).NonOffX=...
    str2num(get(handles.NonOffX,'String'));

% Y Offset of Non-Dominant Hand
function NonOffY_CreateFcn(h, eventdata, handles)
if ispc
    set(h,'BackgroundColor','white');
else
    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function NonOffY_Callback(h, eventdata, handles)
global data
tempstring=strcat(get(h,'String'),'cm');
set(handles.text39,'String',tempstring)
if str2num(get(h,'String'))>0
    tempstring = 'up';
elseif str2num(get(h,'String'))==0
    tempstring = 'No Offset';
else

```

```

    tempstring ='down';

end

set(handles.text45,'String',tempstring);

%stores current value of Non-Dominant Offset Y
data(get(handles.slide_index,'Value')).NonOffY=...
    str2num(get(handles.NonOffY,'String'));

% Z Offset of Non-Dominant Hand
function NonOffZ_CreateFcn(h, eventdata, handles)
if ispc
    set(h,'BackgroundColor','white');
else
    set(h,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function NonOffZ_Callback(h, eventdata, handles)
global data
tempstring=strcat(get(h,'String'),'cm');
set(handles.text40,'String',tempstring)
if str2num(get(h,'String'))>0
    tempstring = 'forward';
elseif str2num(get(h,'String'))==0
    tempstring = 'No Offset';
else

```

```
tempstring = 'backward';  
  
end  
  
set(handles.text46, 'String', tempstring);  
  
%stores current value of Non-Dominant Offset Z  
data(get(handles.slide_index, 'Value')).NonOffZ = ...  
    str2num(get(handles.NonOffZ, 'String'));
```

APPENDIX C

PYTHON CODE

C.1 ASLJack.py

```
# ASLJack.py is base file for signing.

# Creates male signer

#adds aslpy to the search path

sys.path.append('/Python23/aslpy')

#changes working directory to /Python23/aslpy

os.chdir('/Python23/aslpy')

letters = ReadHandShapeFile("/Program
Files/jack40/library/data/sys/JASignedEnglishHandshapes.data")

# creates human figure, call it jack, and creates handle 'j'

#creates jack and all cubes in scene

j=CreateHuman('jack')

execfile('CreateScene.py')

#stores Dictionary words in workspace

execfile('Dictionary.py')
```

```
list = 'Words Available Are: '
for n in range(len(Dictionary)):
    list = list + Dictionary[n] + ' '
#time used in duration for action.py
time = 1.25

#re-assigns jackcore objects to javascript ojects (names)
jack=j
j=jack
tojill='no'
tojack='no'

#jack.AddPosture('stand_normal.post', 'stand_normal')
jackstatus = 'go'
Sentence = raw_input("\nEnter Sentence <type COMMANDS to see special functions>")
if Sentence == 'QUIT':
    jackstatus = 'stop'
elif Sentence == 'JILL':
    tojill='yes'
elif Sentence == 'JACK':
    tojack='yes'

if tojack == 'no':
```



```

if tojill == 'no':
    while jackstatus == 'go':
        GoalDestroy = 'yes'
        print ' '+Sentence
        if Sentence !='QUIT':
            nonletnum = "$!@#%&*&*;,;+_-~='><.&?^[]{}()'"
            transtable = string.maketrans(" , ")
            Sentence = string.translate(Sentence,transtable,nonletnum)

        if Sentence == 'DICTIONARY':
            print Dictionary

        elif Sentence == 'COMMANDS':
            print "\n<type QUIT to exit, DICTIONARY to view word list,JILL for female
signer,JACK for male signer,SPEED to change signing speed>"

        elif Sentence == 'JILL':
            jackstatus='stop'
            tojill='yes'

        elif Sentence == 'JACK':
            jackstatus = 'stop'
            tojack='yes'

        elif Sentence == 'SPEED':
            print("Type speed level (1-10, current speed is ")
            print(-4*(time-2.5)+1)

```

```
level = float(raw_input(""))
```

```
time = 2.5-(level-1)*.25
```

```
else:
```

```
lowWords = Sentence.lower().split()
```

```
for i in range(len(lowWords)):
```

```
    if Dictionary.count(lowWords[i]):
```

```
        # action
```

```
        execfile('action.py')
```

```
    else:
```

```
        execfile('SignLetter.py')
```

```
Sentence = raw_input("Enter Sentence <type COMMANDS to see special  
functions>")
```

```
if Sentence == 'QUIT':
```

```
    jackstatus = 'stop'
```

```
elif Sentence == 'JILL':
```

```
    jackstatus = 'stop'
```

```
    tojill = 'yes'
```

```
elif Sentence == 'JACK':
```

```
    jackstatus = 'stop'
```

```
tojack = 'yes'
```

```
if tojill=='yes':
```

```
    execfile('Destroy.py')
```

```
    execfile('ASLJill.py')
```

```
elif tojack=='yes':
```

```
    execfile('Destroy.py')
```

```
    execfile('ASLJack.py')
```

```
else:
```

```
    print '    Thank you'
```

```
    execfile('Destroy.py')
```

C.2 ASLJill.py

```
# ASLJill.py is base file for signing.

# Creates female signer

#adds aslpy to the search path

sys.path.append('/Python23/aslpy')

#changes working directory to /Python23/aslpy

os.chdir('/Python23/aslpy')

letters = ReadHandShapeFile("/Program
Files/jack40/library/data/sys/JASignedEnglishHandshapes.data")

# creates human figure, call it jack, and creates handle 'j'

#creates jill and all cubes in scene

j=CreateFemale('jill')

execfile('CreateScene.py')

#stores Dictionary words in workspace

execfile('Dictionary.py')

list = 'Words Available Are: '

for n in range(len(Dictionary)):
```

```
list = list + Dictionary[n] + ''

#time used in duration for action.py

time = 1.25

#re-assigns jackcore objects to javascript objects (names)

jack=j

j=jack

tojill='no'

tojack='no'

#jack.AddPosture('stand_normal.post', 'stand_normal')

jackstatus = 'go'

Sentence = raw_input("\nEnter Sentence <type COMMANDS to see special functions>")

if Sentence == 'QUIT':

    jackstatus = 'stop'

elif Sentence == 'JILL':

    tojill='yes'

elif Sentence == 'JACK':

    tojack='yes'

if tojack == 'no':

if tojill == 'no':

    while jackstatus == 'go':
```

```

GoalDestroy = 'yes'

print ' '+Sentence

if Sentence !='QUIT':

    nonletnum ="$!@#$$%^&* ,;+_-~='><.?[[]{}()""

    transtable = string.maketrans("", "")

    Sentence = string.translate(Sentence,transtable,nonletnum)

if Sentence == 'DICTIONARY':

    print Dictionary

elif Sentence == 'COMMANDS':

    print "\n<type QUIT to exit, DICTIONARY to view word list,JILL for female
signer,JACK for male signer,SPEED to change signing speed>"

elif Sentence == 'JILL':

    jackstatus='stop'

    tojill='yes'

elif Sentence == 'JACK':

    jackstatus = 'stop'

    tojack='yes'

elif Sentence == 'SPEED':

    print("Type speed level (1-10, current speed is ")

    print(-4*(time-2.5)+1)

    level = float(raw_input(""))

    time = 2.5-(level-1)*.25

```

```
else:

    lowWords = Sentence.lower().split()

    for i in range(len(lowWords)):

        if Dictionary.count(lowWords[i]):

            # action

            execfile('action.py')

        else:

            execfile('SignLetter.py')
```

```
Sentence = raw_input("Enter Sentence <type COMMANDS to see special  
functions>")
```

```
if Sentence == 'QUIT':

    jackstatus = 'stop'

elif Sentence == 'JILL':

    jackstatus = 'stop'

    tojill = 'yes'

elif Sentence == 'JACK':

    jackstatus = 'stop'

    tojack = 'yes'
```

```
if tojill=='yes':  
    execfile('Destroy.py')  
    execfile('ASLJill.py')  
elif tojack=='yes':  
    execfile('Destroy.py')  
    execfile('ASLJack.py')  
else:  
    print '    Thank you'  
    execfile('Destroy.py')
```


C.3 CreateScene.py

```
# CreateScene.py orients camera and creates all needed objects

# Does not create Jack or Jill

v = View()

v.LookFrom(xyz(-0.250591, -0.002582, -0.000661) * trans(-7.099183, 161.167648,
140.153748))

v.LookAt(trans(-6.708587, 123.651573, -6.409506))

# scl is scale to reduce blocks beyond visibility

scl=.0001

# locations and end effectors

r_orient = CreateFigure("cube.pss","r_orient")

r_orient.Scale(scl)

r_orient.MoveTo(j.segment.right_palm.f11)

r_orient.AttachTo(j.segment.right_palm.f11)

l_orient = CreateFigure("cube.pss","l_orient")

l_orient.Scale(scl)

l_orient.MoveTo(j.segment.left_palm.f11)

l_orient.AttachTo(j.segment.left_palm.f11)
```

```
r_index=CreateFigure("cube.pss","r_index")
r_index.Scale(scl)
r_index.MoveTo(j.segment.right_finger02.base0)
r_index.AttachTo(j.segment.right_finger02.base0)

r_middle=CreateFigure("cube.pss","r_middle")
r_middle.Scale(scl)
r_middle.MoveTo(j.segment.right_finger12.base0)
r_middle.AttachTo(j.segment.right_finger12.base0)

r_ring=CreateFigure("cube.pss","r_ring")
r_ring.Scale(scl)
r_ring.MoveTo(j.segment.right_finger22.base0)
r_ring.AttachTo(j.segment.right_finger22.base0)

r_pinky=CreateFigure("cube.pss","r_pinky")
r_pinky.Scale(scl)
r_pinky.MoveTo(j.segment.right_finger32.base0)
r_pinky.AttachTo(j.segment.right_finger32.base0)

l_index=CreateFigure("cube.pss","l_index")
l_index.Scale(scl)
l_index.MoveTo(j.segment.left_finger02.base0)
```

```
l_index.AttachTo(j.segment.left_finger02.base0)
```

```
l_middle=CreateFigure("cube.pss","l_middle")
```

```
l_middle.Scale(scl)
```

```
l_middle.MoveTo(j.segment.left_finger12.base0)
```

```
l_middle.AttachTo(j.segment.left_finger12.base0)
```

```
l_ring=CreateFigure("cube.pss","l_ring")
```

```
l_ring.Scale(scl)
```

```
l_ring.MoveTo(j.segment.left_finger22.base0)
```

```
l_ring.AttachTo(j.segment.left_finger22.base0)
```

```
l_pinky=CreateFigure("cube.pss","l_pinky")
```

```
l_pinky.Scale(scl)
```

```
l_pinky.MoveTo(j.segment.left_finger32.base0)
```

```
l_pinky.AttachTo(j.segment.left_finger32.base0)
```

```
l_thumb=CreateFigure("cube.pss","l_thumb")
```

```
l_thumb.Scale(scl)
```

```
l_thumb.MoveTo(j.segment.left_thumb2.base0)
```

```
l_thumb.AttachTo(j.segment.left_thumb2.base0)
```

```
r_thumb=CreateFigure("cube.pss","r_thumb")
```

```
r_thumb.Scale(scl)
r_thumb.MoveTo(j.segment.right_thumb2.base0)
r_thumb.AttachTo(j.segment.right_thumb2.base0)

l_thumb=CreateFigure("cube.pss","cube_l_thumb")
l_thumb.Scale(scl)
l_thumb.MoveTo(j.segment.left_thumb2.tip)
l_thumb.AttachTo(j.segment.left_thumb2.tip)

l_cpalm=CreateFigure("cube.pss","l_cpalm")
l_cpalm.Scale(scl)
l_cpalm.MoveTo(j.segment.left_palm.palmcenter)
l_cpalm.AttachTo(j.segment.left_palm.palmcenter)

r_cpalm=CreateFigure("cube.pss","r_cpalm")
r_cpalm.Scale(scl)
r_cpalm.MoveTo(j.segment.right_palm.palmcenter)
r_cpalm.AttachTo(j.segment.right_palm.palmcenter)

# only locations
cube_r_bpalm=CreateFigure("cube.pss","cube_r_bpalm")
cube_r_bpalm.Scale(scl)
cube_r_bpalm.MoveTo(j.segment.right_palm.front)
```

```
cube_r_bpalm.AttachTo(j.segment.right_palm.front)
```

```
#cubes moving around to be goal objects
```

```
DomGoal = CreateFigure("cube.pss","DomGoal")
```

```
DomGoal.Scale(scl)
```

```
NonGoal = CreateFigure("cube.pss","NonGoal")
```

```
NonGoal.Scale(scl)
```

```
cube_r_ear=CreateFigure("cube.pss","cube_r_ear")
```

```
cube_r_ear.Scale(scl)
```

```
cube_r_ear.MoveTo(j.segment.bottom_head.right)
```

```
cube_r_ear.AttachTo(j.segment.bottom_head.right)
```

```
cube_l_ear=CreateFigure("cube.pss","cube_l_ear")
```

```
cube_l_ear.Scale(scl)
```

```
cube_l_ear.MoveTo(j.segment.bottom_head.left)
```

```
cube_l_ear.AttachTo(j.segment.bottom_head.left)
```

```
cube_r_eye=CreateFigure("cube.pss","cube_r_eye")
```

```
cube_r_eye.Scale(scl)
```

```
cube_r_eye.MoveTo(j.segment.bottom_head.right_eyeball)
```

```
cube_r_eye.AttachTo(j.segment.bottom_head.right_eyeball)
```

```
cube_l_eye=CreateFigure("cube.pss","cube_l_eye")
cube_l_eye.Scale(scl)
cube_l_eye.MoveTo(j.segment.bottom_head.left_eyeball)
cube_l_eye.AttachTo(j.segment.bottom_head.left_eyeball)

cube_m_chin=CreateFigure("cube.pss","cube_chin")
cube_m_chin.Scale(scl)
cube_m_chin.MoveTo(j.segment.bottom_head.menton)
cube_m_chin.AttachTo(j.segment.bottom_head.menton)

cube_r_bicep=CreateFigure("cube.pss","cube_r_bicep")
cube_r_bicep.Scale(scl)
cube_r_bicep.MoveTo(j.segment.right_upper_arm.front)
cube_r_bicep.AttachTo(j.segment.right_upper_arm.front)

cube_r_elbow=CreateFigure("cube.pss","cube_r_elbow")
cube_r_elbow.Scale(scl)
cube_r_elbow.MoveTo(j.segment.right_lower_arm.front)
cube_r_elbow.AttachTo(j.segment.right_lower_arm.front)

cube_l_elbow=CreateFigure("cube.pss","cube_l_elbow")
cube_l_elbow.Scale(scl)
```

```
cube_l_elbow.MoveTo(j.segment.left_upper_arm.distal)
cube_l_elbow.AttachTo(j.segment.left_upper_arm.distal)

cube_r_wrist=CreateFigure("cube.pss","cube_r_wrist")
cube_r_wrist.Scale(scl)
cube_r_wrist.MoveTo(j.segment.right_lower_arm.distal)
cube_r_wrist.AttachTo(j.segment.right_lower_arm.distal)

cube_l_wrist=CreateFigure("cube.pss","cube_l_wrist")
cube_l_wrist.Scale(scl)
cube_l_wrist.MoveTo(j.segment.left_lower_arm.distal)
cube_l_wrist.AttachTo(j.segment.left_lower_arm.distal)

cube_neck=CreateFigure("cube.pss","cube_neck")
cube_neck.Scale(scl)
cube_neck.MoveTo(j.segment.neck.front)
cube_neck.AttachTo(j.segment.neck.front)

# from
cube_r_hip=CreateFigure("cube.pss","cube_r_hip")
cube_r_hip.Scale(scl)
cube_r_hip.MoveTo(j.segment.l1.distal, offset=(-12, -20, 15))
```

```
cube_l_hip=CreateFigure("cube.pss","cube_l_hip")
```

```
cube_l_hip.Scale(scl)
```

```
cube_l_hip.MoveTo(j.segment.l1.distal, offset=(12, -20, 15))
```

```
cube_hip=CreateFigure("cube.pss","cube_hip")
```

```
cube_hip.Scale(scl)
```

```
cube_hip.MoveTo(j.segment.l1.distal, offset=(0, -20, 15))
```

```
cube_r_shoulder=CreateFigure("cube.pss","cube_r_shoulder")
```

```
cube_r_shoulder.Scale(scl)
```

```
cube_r_shoulder.MoveTo(j.segment.l1.distal, offset=(-12, 20, 12))
```

```
cube_l_shoulder=CreateFigure("cube.pss","cube_l_shoulder")
```

```
cube_l_shoulder.Scale(scl)
```

```
cube_l_shoulder.MoveTo(j.segment.l1.distal, offset=(12, 20, 12))
```

```
cube_shoulder=CreateFigure("cube.pss","cube_shoulder")
```

```
cube_shoulder.Scale(scl)
```

```
cube_shoulder.MoveTo(j.segment.l1.distal, offset=(0, 20, 12))
```

```
cube_chest=CreateFigure("cube.pss","cube_chest")
```

```
cube_chest.Scale(scl)
```

```
cube_chest.MoveTo(j.segment.l1.distal, offset=(0, 5, 15))
```



```
cube_r_chest=CreateFigure("cube.pss","cube_r_chest")
```

```
cube_r_chest.Scale(scl)
```

```
cube_r_chest.MoveTo(j.segment.l1.distal, offset=(-12, 5, 15))
```

```
cube_l_chest=CreateFigure("cube.pss","cube_l_chest")
```

```
cube_l_chest.Scale(scl)
```

```
cube_l_chest.MoveTo(j.segment.l1.distal, offset=(12, 5, 15))
```

```
cube_nose=CreateFigure("cube.pss","cube_nose")
```

```
cube_nose.Scale(scl)
```

```
cube_nose.MoveTo(j.segment.bottom_head.sight, offset=(0, -4, 3))
```

```
cube_m_mouth=CreateFigure("cube.pss","cube_m_mouth")
```

```
cube_m_mouth.Scale(scl)
```

```
cube_m_mouth.MoveTo(j.segment.bottom_head.sight, offset=(0, -8, 0))
```

```
cube_r_mouth=CreateFigure("cube.pss","cube_r_mouth")
```

```
cube_r_mouth.Scale(scl)
```

```
cube_r_mouth.MoveTo(j.segment.bottom_head.sight, offset=(-2.5, -6, 1))
```

```
cube_l_mouth=CreateFigure("cube.pss","cube_l_mouth")
```

```
cube_l_mouth.Scale(scl)
```

```
cube_l_mouth.MoveTo(j.segment.bottom_head.sight, offset=(2.5, -6, 1))
```

```
cube_l_brow=CreateFigure("cube.pss","cube_l_brow")
```

```
cube_l_brow.Scale(scl)
```

```
cube_l_brow.MoveTo(j.segment.bottom_head.sight, offset=(2.5, 4, 1))
```

```
cube_r_brow=CreateFigure("cube.pss","cube_r_brow")
```

```
cube_r_brow.Scale(scl)
```

```
cube_r_brow.MoveTo(j.segment.bottom_head.sight, offset=(-2.5, 4, 1))
```

```
cube_c_brow=CreateFigure("cube.pss","cube_c_brow")
```

```
cube_c_brow.Scale(scl)
```

```
cube_c_brow.MoveTo(j.segment.bottom_head.sight, offset=(0, 4, 1))
```

```
cube_r_chin=CreateFigure("cube.pss","cube_r_chin")
```

```
cube_r_chin.Scale(scl)
```

```
cube_r_chin.MoveTo(j.segment.bottom_head.sight, offset=(-2.5, -8, 1))
```

```
cube_l_chin=CreateFigure("cube.pss","cube_l_chin")
```

```
cube_l_chin.Scale(scl)
```

```
cube_l_chin.MoveTo(j.segment.bottom_head.sight, offset=(2.5, -8, 1))
```

C.4 Dictionary.py

```
# Dictionary.py initiates categories of letters in lists
# also creates 'Dictionary' which contains all signs available
# this is dynamic and avoids maintaining a sign list

upoutList = 'A B C D E F I K L M N O R S T U V W X Y 0 6 7 8 9'
upoutList = upoutList.split()

upinList = '1 2 3 4 5'
upinList = upinList.split()

pqList = 'P Q'
pqList = pqList.split()

hgList = 'G H'
hgList = hgList.split()

Dictionary = os.listdir('SignData/')
Dictionary.pop(Dictionary.index('upoutLetter.py'))
Dictionary.pop(Dictionary.index('upinLetter.py'))
Dictionary.pop(Dictionary.index('ghLetter.py'))
Dictionary.pop(Dictionary.index('pqLetter.py'))
Dictionary.pop(Dictionary.index('doubleLetter.py'))
Dictionary.pop(Dictionary.index('testgui11.m'))
Dictionary.pop(Dictionary.index('testgui11.asv'))
```

```
Dictionary.pop(Dictionary.index('testgui11.fig'))
```

```
Dictionary.pop(Dictionary.index('picture'))
```

```
Dictionary.pop(Dictionary.index('handshapes'))
```

```
Dictionary.pop(Dictionary.index('locations'))
```

```
Dictionary.pop(Dictionary.index('j.py'))
```

```
Dictionary.pop(Dictionary.index('z.py'))
```

```
for n in range(len(Dictionary)):
```

```
    temp = Dictionary[n]
```

```
    Dictionary[n] = temp[0:len(temp)-3]
```

C.5 action.py

```

# action.py executes movement of hands
# executes files with parameters stored
execfile('SignData/'+lowWords[i] + '.py')

for n in range(number):

    DomEffector[n]=(DomEffector[n],")
    NonEffector[n]=(NonEffector[n],")

    execfile('CubeScene.py')
    execfile('HandScene.py')
    execfile('PositionGoal.py')

    if DomLocale[n] == 'inherited':

DomGoal.SetLocation(xyz(DomGoal.GetLocation().xyz()*trans(DomMove.GetLocatio
n().trans()))

    if NonLocale[n] == 'inherited':

NonGoal.SetLocation(xyz(NonGoal.GetLocation().xyz()*trans(NonMove.GetLocation()
.trans()))

    execfile('HandScene.py')

    Movement=DomSet

    execfile('MovementScene.py')

```

```
DomMove=xyz(DomGoal.GetLocation().xyz()*trans(DomGoal.GetLocation().trans()+
moveoffset[n])
```

```
  Movement=NonSet
```

```
  execfile('MovementScene.py')
```

```
NonMove=xyz(NonGoal.GetLocation().xyz()*trans(NonGoal.GetLocation().trans()+mo
veoffset[n])
```

```
  DoInOrder(DoTogether(jack.Reach(handed, DomGoal, endeff=domeff, poweigh=.2,
duration = time, otype='align_frame'),\
```

```
    HandShapeMotion(jack, letters[DomShape[n]], handed, duration = time),\
```

```
    jack.Reach(nondom, NonGoal, endeff=noneff, poweigh=.2, duration =
time, otype='align_frame'),\
```

```
    HandShapeMotion(jack, letters[NonShape[n]], nondom, duration= time)),\
```

```
    DoTogether(HandShapeMotion(jack, letters[DomShape[n]], handed, duration =time),\
```

```
    HandShapeMotion(jack, letters[NonShape[n]], nondom, duration= time)),\
```

```
    DoTogether(jack.Reach(handed, DomMove, endeff=domeff, poweigh=.2, duration =
time1, otype='align_frame'),\
```

```
    jack.Reach(nondom, NonMove, endeff=noneff, poweigh=.2,
```

```
    duration=time1, otype='align_frame'))))
```

```
  Flush()
```

C.6 CubeScene.py

```
# CubeScene.py stores location value of destination

# Stores value of key location on body

#for Dominant Hand

if DomLocale[n] == 'left ear':

    DomCube = scene.cube_l_ear

elif DomLocale[n] == 'right ear':

    DomCube = scene.cube_r_ear

elif DomLocale[n] == 'chin':

    DomCube = scene.cube_chin

elif DomLocale[n] == 'neck':

    DomCube = scene.cube_neck

elif DomLocale[n] == 'right elbow':

    DomCube = scene.cube_r_elbow

elif DomLocale[n] == 'left elbow':

    DomCube = scene.cube_l_elbow

#elif DomLocale[n] == 'right bicep':

# DomCube = scene.cube_r_bicep

elif DomLocale[n] == 'chest center':

    DomCube = scene.cube_chest

elif DomLocale[n] == 'chest right':
```

```
    DomCube = scene.cube_r_chest
elif DomLocale[n] == 'chest left':
    DomCube = scene.cube_l_chest
elif DomLocale[n] == 'nose':
    DomCube = scene.cube_nose
elif DomLocale[n] == 'forehead':
    DomCube = scene.cube_c_brow
elif DomLocale[n] == 'right brow':
    DomCube = scene.cube_r_brow
elif DomLocale[n] == 'left brow':
    DomCube = scene.cube_l_brow
elif DomLocale[n] == 'right wrist':
    DomCube = scene.cube_r_wrist
elif DomLocale[n] == 'left wrist':
    DomCube = scene.cube_l_wrist
elif DomLocale[n] == 'right hip':
    DomCube = scene.cube_r_hip
elif DomLocale[n] == 'left hip':
    DomCube = scene.cube_l_hip
elif DomLocale[n] == 'hip center':
    DomCube = scene.cube_hip
elif DomLocale[n] == 'right shoulder':
    DomCube = scene.cube_r_shoulder
```



```
elif DomLocale[n] == 'left shoulder':  
    DomCube = scene.cube_l_shoulder  
elif DomLocale[n] == 'shoulder center':  
    DomCube = scene.cube_shoulder  
elif DomLocale[n] == 'right first knuckle':  
    DomCube = scene.r_orient  
elif DomLocale[n] == 'left first knuckle':  
    DomCube = scene.l_orient  
elif DomLocale[n] == 'right index finger':  
    DomCube = scene.r_index  
elif DomLocale[n] == 'left index finger':  
    DomCube = scene.l_index  
elif DomLocale[n] == 'right middle finger':  
    DomCube = scene.r_middle  
elif DomLocale[n] == 'left middle finger':  
    DomCube = scene.l_middle  
elif DomLocale[n] == 'right ring finger':  
    DomCube = scene.r_ring  
elif DomLocale[n] == 'left ring finger':  
    DomCube = scene.l_ring  
elif DomLocale[n] == 'right pinky finger':  
    DomCube = scene.r_pinky  
elif DomLocale[n] == 'left pinky finger':
```

```
    DomCube = scene.l_pinky
elif DomLocale[n] == 'right thumb':
    DomCube = scene.r_thumb
elif DomLocale[n] == 'left thumb':
    DomCube = scene.l_thumb
elif DomLocale[n] == 'right palm center':
    DomCube = scene.r_cpalm
elif DomLocale[n] == 'left palm center':
    DomCube = scene.l_cpalm
else:
    DomCube = scene.cube_chest
```

C.7 PositionalGoal.py

```

# PositionalGoal.py rotates destination block to proper orientation.
# PositionalGoal.py only rotates, does not translate.

RYL = 180 # rotation initial condition for left hand
RYR = 0    # rotation initial condition for right hand
if handed == 'right':
    if NonEffector[n][0] == 'palm center':
        RYL = 0
        NonX[n] = -NonX[n]
        NonZ[n] = -NonZ[n]
        RYD = RYR
        RYN = RYL
    else:
        if DomEffector[n][0] == 'palm center':
            RYL = 0
            RYD = RYL
            RYN = RYR

#orients goal cubes to same orientation as palm down, pointing forward
DomGoal.SetLocation(xyz(90*u.deg,RYD*u.deg,0*u.deg)*trans(DomCube.GetLocation
().trans()+[DomOffX[n],DomOffY[n],DomOffZ[n]]))

```

```
NonGoal.SetLocation(xyz(90*u.deg,RYN*u.deg,0*u.deg)*trans(NonCube.GetLocation()
.trans()+[NonOffX[n],NonOffY[n],NonOffZ[n]]))
```

```
Flush()
```

```
if 1:
```

```
DoInOrder(DoTogether(DomGoal.Move(xyz(0*u.deg,0*u.deg,DomX[n]*u.deg),duration
= 0,asSeenBy = scene.DomGoal),\
```

```
NonGoal.Move(xyz(0*u.deg,0*u.deg,NonX[n]*u.deg),duration = 0,asSeenBy =
scene.NonGoal)),\
```

```
DoTogether(DomGoal.Move(xyz(DomZ[n]*u.deg,0*u.deg,0*u.deg),duration =
0,asSeenBy = scene.DomGoal),\
```

```
NonGoal.Move(xyz(NonZ[n]*u.deg,0*u.deg,0*u.deg),duration = 0,asSeenBy =
scene.NonGoal)),\
```

```
DoTogether(DomGoal.Move(xyz(0*u.deg,DomY[n]*u.deg,0*u.deg),duration =
0,asSeenBy = scene.DomGoal),\
```

```
NonGoal.Move(xyz(0*u.deg,NonY[n]*u.deg,0*u.deg),duration = 0,asSeenBy =
scene.NonGoal)))
```

```
Flush()
```

C.8 MovementScene.py

```
# MovementScene.py incorporates the linear movement parameters by
# calculating an offset value reflecting the end position relative to
# the start position.

moveoffset = [(0,0,0)]

for m in range(number-1):
    moveoffset.append((0,0,0))

if len(Movement[n][0])==1:
    Movement[n]=(Movement[n],")

#Looks for string in list 'Movement'
#then alters the movement offset

if 'up' in Movement[n]:
    moveoffset[n] = (moveoffset[n][0],moveoffset[n][1]+20,moveoffset[n][2])
    time1=time

if 'down' in Movement[n]:
    moveoffset[n] = (moveoffset[n][0],moveoffset[n][1]-20,moveoffset[n][2])
    time1=time
```

if 'left' in Movement[n]:

 moveoffset[n] = (moveoffset[n][0]+20,moveoffset[n][1],moveoffset[n][2])

 time1=time

if 'right' in Movement[n]:

 moveoffset[n] = (moveoffset[n][0]-20,moveoffset[n][1],moveoffset[n][2])

 time1=time

if 'away from Jack' in Movement[n]:

 moveoffset[n] = (moveoffset[n][0],moveoffset[n][1],moveoffset[n][2]+20)

 time1=time

if 'toward Jack' in Movement[n]:

 moveoffset[n] = (moveoffset[n][0],moveoffset[n][1],moveoffset[n][2]-20)

 time1=time

if 'stationary' in Movement[n]:

 moveoffset[n] = (0,0,0)

 time1=0

C.9 HandScene.py

```
# HandScene.py designates the end effector for each hand

if handed == 'left':

    if DomEffector[n][0] == "first knuckle":

        domeff = scene.l_orient.l_orient.base

    elif DomEffector[n][0] == "index finger":

        domeff = scene.l_index.l_index.base

    elif DomEffector[n][0] == "middle finger":

        domeff = scene.l_middle.l_middle.base

    elif DomEffector[n][0] == "ring finger":

        domeff = scene.l_ring.l_ring.base

    elif DomEffector[n][0] == "pinky finger":

        domeff = scene.l_pinky.l_pinky.base

    elif DomEffector[n][0] == "thumb":

        domeff = scene.l_thumb.l_thumb.base

    elif DomEffector[n][0] == "palm center":

        domeff = scene.l_cpalm.l_cpalm.base

else:

    domeff = scene.l_orient.l_orient.base

if NonEffector[n][0] == "first knuckle":

    noneff = scene.r_orient.r_orient.base

elif NonEffector[n][0] == "index finger":

    noneff = scene.r_index.r_index.base
```

```
elif NonEffector[n][0] == "middle finger":
    noneff = scene.r_middle.r_middle.base
elif NonEffector[n][0] == "ring finger":
    noneff = scene.r_ring.r_ring.base
elif NonEffector[n][0] == "pinky finger":
    noneff = scene.r_pinky.r_pinky.base
elif NonEffector[n][0] == "thumb":
    noneff = scene.r_thumb.r_thumb.base
elif NonEffector[n][0] == "palm center":
    noneff = scene.r_cpalm.r_cpalm.base
else:
    noneff = scene.r_orient.r_orient.base
else:
    if DomEffector[n] == "first knuckle":
        domeff = scene.r_orient.r_orient.base
    elif DomEffector[n][0] == "index finger":
        domeff = scene.r_index.r_index.base
    elif DomEffector[n][0] == "middle finger":
        domeff = scene.r_middle.r_middle.base
    elif DomEffector[n][0] == "ring finger":
        domeff = scene.r_ring.r_ring.base
    elif DomEffector[n][0] == "pinky finger":
        domeff = scene.r_pinky.r_pinky.base
```



```
elif DomEffector[n][0] == "thumb":  
    domeff = scene.r_thumb.r_thumb.base  
elif DomEffector[n][0] == "palm center":  
    domeff = scene.r_cpalm.r_cpalm.base  
else:  
    domeff = scene.r_orient.r_orient.base  
  
if NonEffector[n][0] == "first knuckle":  
    noneff = scene.l_orient.l_orient.base  
elif NonEffector[n][0] == "index finger":  
    noneff = scene.l_index.l_index.base  
elif NonEffector[n][0] == "middle finger":  
    noneff = scene.l_middle.l_middle.base  
elif NonEffector[n][0] == "ring finger":  
    noneff = scene.l_ring.l_ring.base  
elif NonEffector[n][0] == "pinky finger":  
    noneff = scene.l_pinky.l_pinky.base  
elif NonEffector[n][0] == "thumb":  
    noneff = scene.l_thumb.l_thumb.base  
elif NonEffector[n][0] == "palm center":  
    noneff = scene.l_cpalm.l_cpalm.base  
else:  
noneff = scene.l_orient.l_orient.base
```

C.10 SignLetter.py

```

# SignLetter.py fingerspells words not in the dictionary

# special features include double letters, and double numbers

jack._CachePostures()

n = 0  ## Put the variable 'n'in in-order to be able to use the subsequent letters in
conditional statements, eg.(1&0 is ten)

preletter='$'

# lowWords is a word in the sentence

wordi = lowWords[i]

Doubles = []

for n in range(len(wordi)-1):

    Doubles = Doubles + []

# This eliminates double letters and saves their position in the word

wordi = wordi.upper()

word=wordi

wordi=""

n=0

nn=0

while n < len(word)-1:

    wordi = wordi+word[n]

    if word[n]==word[n+1]:

```

```

n=n+1

Doubles[nn]=len(wordi)-1

nn=nn+1

n=n+1

if n == len(word)-1:

    wordi = wordi+word[n]

# Destroys empty compartments in Doubles

for n in range(len(Doubles)-1,0, -1):

    if Doubles[n] == "":

        Doubles.pop(n)

n=0

for n in range(len(wordi)):

    letter = wordi[n]

    if letter in upoutList:

        execfile('SignData/upoutLetter.py')

    elif letter in upinList:

        execfile('SignData/upinLetter.py')

    elif letter in pqList:

        execfile('SignData/pqLetter.py')

    elif letter in hgList:

        execfile('SignData/ghLetter.py')

    elif letter == 'j':

```

```
    execfile('SignData/j.py')  
  
elif letter == 'z':  
  
    execfile('SignData/z.py')  
  
  
  
# designates correct handles for left and right hand  
execfile('HandScene.py')  
  
# double numbers are signed palm down  
if n in Doubles:  
  
    if wordi[n] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']:  
  
        execfile('SignData/pqLetter.py')  
  
        DomSet[0] = [('right')]  
  
# action  
execfile('action.py')
```

C.11 Destroy.py

Destroy.py deletes all cubes and Jack/Jill.

Destroy(j)

Destroy(r_orient)

Destroy(l_orient)

Destroy(r_index)

Destroy(l_index)

Destroy(r_middle)

Destroy(l_middle)

Destroy(r_ring)

Destroy(l_ring)

Destroy(r_pinky)

Destroy(l_pinky)

Destroy(r_thumb)

Destroy(l_thumb)

Destroy(l_cpalm)

Destroy(r_cpalm)

Destroy(DomGoal)

Destroy(NonGoal)

Destroy(cube_r_ear)

Destroy(cube_l_ear)

Destroy(cube_r_eye)

Destroy(cube_l_eye)

Destroy(cube_m_chin)

Destroy(cube_r_bicep)

Destroy(cube_r_elbow)

Destroy(cube_l_elbow)

Destroy(cube_r_wrist)

Destroy(cube_l_wrist)

Destroy(cube_neck)

Destroy(cube_r_bpalm)

Destroy(cube_r_hip)

Destroy(cube_l_hip)

Destroy(cube_hip)

Destroy(cube_r_shoulder)

Destroy(cube_l_shoulder)

Destroy(cube_shoulder)

Destroy(cube_chest)

Destroy(cube_r_chest)

Destroy(cube_l_chest)

Destroy(cube_nose)

Destroy(cube_m_mouth)

Destroy(cube_r_mouth)

Destroy(cube_l_mouth)

Destroy(cube_l_brow)

Destroy(cube_r_brow)

Destroy(cube_c_brow)

Destroy(cube_r_chin)

Destroy(cube_l_chin)

C.12 airplane.py

```
# airplane

number = 2

nondom = 'left'

handed = 'right'

sign_name = 'airplane'

DomEffector = ['first knuckle','first knuckle']

NonEffector = ['first knuckle','first knuckle']

DomShape = ['Plane','Plane']

NonShape = ['rest','rest']

DomLocale = ['right chest','left shoulder']

NonLocale = ['left hip','left hip']

DomX = [0,0]

DomY = [0,0]

DomZ = [0,-45]

NonX = [0,0]

NonY = [-90,-90]

NonZ = [-90,-90]

DomOffX = [-15,0]

DomOffY = [0,0]

DomOffZ = [20,20]
```


NonOffX = [10,10]

NonOffY = [-30,-30]

NonOffZ = [0,0]

DomSet = [('stationary'),('stationary')]

NonSet = [('stationary'),('stationary')]

REFERENCES

- [1] *Jack 2.3 Training Manual*. Engineering Animation Inc., 1999.
- [2] *General Python FAQ*, [Online document], [cited 2004 Jun 10], Available HTTP: <http://www.python.org/doc/faq/general.html>.
- [3] What Is The Difference Between a Deaf and a Hard of Hearing Person? [Online document], [cited 2004 Aug 12], Available HTTP: <http://pages.ivillage.com/cl-loluv/id8.html>.
- [4] *Etiologies and Causes of Deafness*, [Online document], [cited 2004 Aug 12], Available HTTP: <http://library.gallaudet.edu/dr/faq-etiol.html>.
- [5] *Literacy and Deaf Students*, [Online document], [cited 2004 Aug 12], Available HTTP: <http://gri.gallaudet.edu/Literacy/>.
- [6] S. Foster and J. MacLeod, *Deaf People at Work: Assessment of Communication Among the Deaf and Hearing in Work Settings*. International Journal of Audiology Jul 2003 Supplement 1, Vol 42, pS128, 12p.
- [7] *American Sign Language History: History of Sign Language*, [Online document], [cited 2004 Jul 16], Available HTTP: <http://www.westislandlife.com/asl/history.htm>.
- [8] P. Gabel, Personal Communications, Sign Language Center, New York City, NY, [2004 Jun-Aug].
- [9] W. C. Stokoe, *A Dictionary of American Sign Language on Linguistic Principles, New Edition*. Linstock Press, Silver Spring, MD 1976.
- [10] E. E. Schneider, *American Sign Language (ASL) vs. Signed English (SE)*, [Online document], April 28, 2001, [cited 2004 May 20], Available HTTP: <http://www.lesstutor.com/eesASLIntro.html>.
- [11] J. Berke, *Signed Exact English*, [Online document], [cited 2004 Jul 16], Available HTTP: <http://deafness.about.com/cs/signfeats2/a/signedenglish.htm>.
- [12] E. Costello, Ph.D., *Random House Webster's Concise American Sign Language Dictionary*, Bandom Books, New York, January 2002.
- [13] T. Humphries and C. Padden, *Learning American Sign Language*, Prentice Hall, Englewood Cliffs, NJ, 1992.
- [14] L. G. Lane, *Gallaudet Survival Guide to Signing*, Gallaudet University Press, Washington D.C., 1990.

- [15] M. Sternberg, *The American Sign Language Dictionary on CD-ROM*. Harper Collins, NY, 1994.
- [16] F. Solina, and S. Krapez. *Multimedia Dictionary and Synthesis of Sign Language*, [Online document], [cited 2004 Apr 11], Available HTTP: <http://eprints.fri.uni-lj.si/archive/00000039/01/MMdict2001.pdf>.
- [17] M. A. Mandel Ph.D., *ASCII-Stokoe Notation: A Computer-Writeable Transliteration System for Stokoe Notation of American Sign Language*, [Online document], Draft 1993.03.23 [cited 2004 Apr 11], Available HTTP: <http://world.std.com/~mam/ASCII-Stokoe.html>.
- [18] *DePaul University Develop CPU Program That Translates Spoken English to ASL*, [Online document], CHICAGO, Aug 7, 2002 (BUSINESS WIRE). [cited 2004 Jul 7], Available HTTP: <http://deafbase.com/modules.php?name=News&file=print&sid=258>.
- [19] *Signing Avatars*, [Online document], August 29th 2002, [cited 2004 Aug 11], Available HTTP: http://www.bbcworld.com/content/clickonline_archive_35_2002.asp?pageid=666&co_pageid=3.
- [20] A. B. Grieve-Smith, *A Demonstration of Text-to-Sign Synthesis*, [Online document], Presented at the Fourth Workshop on Gesture and Human-Computer Interaction, London 2001, [cited 2004 Mar 15], Available HTTP: <http://www.unm.edu/~grvsmth/signsynth/gw2001/>.
- [21] A. B. Grieve-Smith, *Sign Synthesis and Sign Phonology*, [Online document], Linguistics Department, University of New Mexico. Published in *Proceedings of the First High Desert Student Conference in Linguistics*, 1998, [cited 2004 Mar 15], Available HTTP: <http://www.unm.edu/~grvsmth/portfolio/sssp.pdf>.
- [22] A. B. Grieve-Smith, *Welcome to SignSynth*, [Online document], [cited 2004 Mar 15], Available HTTP: <http://www.unm.edu/~grvsmth/signsynth/gw2001/ascsto.html> .
- [23] J. Allen, *Parametric Synthesis of American Sign Language*, presented to the staff of New Jersey Institute of Technology, May 2004.
- [24] *ASL Dictionary*, [Online document], [cited 2004 Jul 18], Available HTTP: http://library.thinkquest.org/10202/asl_dictionary_text.html?tqskip1=1
- [25] *American Sign Language Browser*, ©2000 Michigan State University Communication Technology Laboratory, [Online document], [cited 2004 Jul 18], Available HTTP: <http://commtechlab.msu.edu/sites/aslweb/browser.htm>

- [26] G. B. Sherry, *A Kinematic Analysis of Hand Configurations in Static and Dynamic Fingerspelling*, presented to the staff of New Jersey Institute of Technology, May 2004.